Code Composer Studio Help

Contents

Introduction to Code Composer

Code Composer Studio is an Integrated Development Environment for Texas Instruments processors that is based on the Eclipse open source software framework. Eclipse content has not been modified thus it supports integration with many different plugins. Code Composer streamlines the design process with a more intuitive interface combined with leading C code density and powerful debugging capabilities. Major upgrades include optimizations and improvements focused on improving design flow, simplifying debugging and verification and speeding time-to-market while keeping system costs low.

Reference Areas

- Key New Code Composer Features
- Where to Begin
- DSP Knowledgebase

- Technical Support

Key New Features

New functionality Code Composer Studio includes:

- **Emulation Drivers**—Included some third-party emulation drivers in the installation. Essentially those emulators that are sold through the TI eStore will have their drivers on the CD to make things easier for initial installation. The drivers for DSKs are also present.
- **Migrating Existing Projects**—CCSv6 is a major release and as a result it is quite different from previous versions. Therefore there are some migration steps that you need to go through to move your development from previous versions to v4. There are some tools to help you, such as the project import wizard. The overall environment is much different so you will need to patiently adjust to it.
  - **CCS v3.3 Projects**—CCSv5 uses the Eclipse concept of projects. To help you import your existing projects into CCSv5 we have created a project import wizard. Use the Project > Import Legacy CCSv3.3 Project menu command. This starts a wizard that guides you through the conversion process. If you don't want to convert your projects at this time, you may want to create a standard make project that calls `timake` to build your existing project file.
  - **CCE v3 Projects**—CCSv4 and CCEv3.x projects are very similar. Please use Project -> Import Existing CC/CCE Eclipse projects menu.
  - **CCS v4 Projects**—CCSv4 projects can be imported and are automatically converted to newer format. Please use Project -> Import Existing CCS/CCE Eclipse projects menu.
  - **CCS v5 Projects**—CCSv5 projects can be imported and are automatically converted to newer format. Please use Project -> Import Existing CCS/CCE Eclipse projects menu.
- **Target Configuration and Setup Editor**—New target configuration files may be created from File -> New -> Target Configuration menu. If you have a project selected in Project Explorer view, then target configuration wizard will use that project as the location where to create this file. User may adjust it. Alternatively, target configuration files are created in default location. If target configuration file is added to the project, then it will be automatically used by the debugger when the starting a debugger action is selected. Setup Editor allows users to create target configuration files using basic tab where simplified selection are available (i.e. select connection and board/device). Setup Editor may also be used to configure target settings from scratch using Advanced tab in Setup Editor. One of these configurations can be designated as your *Default Target Configuration* using Target Configuration View (View->Target Configuration)
- **Real-time Analysis**—transfers DSP/BIOS (and user-defined) logs to provide a pattern of a processor's kernel task execution and resource usage. Once the logs are transferred from the target to the host, you can display graphs of Task Interaction over time, CPU Usage, CPU Load on a Task basis, and the System Execution States. A control panel helps you select which logs you want to collect, transfer, and display.
- **Profiling and Code Coverage**—perform function-level and source line (within a function) profiling. User can sort functions by name or percentage of time usage in the display viewer. Profiling data from multiple runs can be displayed. Displays produced include Function Level Profile, Application Level Profiling, and Code Coverage analysis tables.
- **Memory View changes**—In addition to displaying memory, some CPUs have multi-hierarchical memory that can be examined using memory view. That is, inspect the contents of the L1P cache, L1D, L2, and so forth. On devices that support MMU (Memory Management Units, such as OMAP35xx) users need to display and edit virtual memory data. For devices that support multiple memory pages, users need to edit and display memory on different pages. Memory view permits the user to configure the data format, precision, endianness, and data type appropriate for your application.
- **GEL (General Extension Language)**—GEL is still present in CCS. GEL is the expression evaluator used by the CCS debugger. Each location where you enter in a start address, variable name, or condition on a breakpoint it is using GEL to evaluation that expression. *(Changes)* In CCSv4 GEL has changed to be part of the CCS debugger and not the top level user interface, you can not run some of the old GEL functions that accessed the user interface such as GEL_WatchAdd() or the project related functions. Scripting Console may be used to provide most of this functionality that requires interaction with Graphical User Interface. However debug related functions still work. Hotmenu items defined in GEL scripts are visible under "Scripts" menu, they are visible after debugger is started and are sensitive to processor selected in Debug View. i.e. in multi-processor scenario to see ARM GEL menu's user would need to select ARM processor in Debug View.
- **Increased ease-of-use**—streamlined options dialog boxes reduce the number of steps required for commonly performed design tasks.
- **High-performance debugger**—replaces the GDB debugger with the TI Debug Server yields up to 200 percent faster performance for tasks such as downloading applications, single-stepping, and refreshing registers and variables.
- **Updated C-code syntax**—helps developers to directly import code without modification, from the rich diversity of code examples, libraries and demos from TI and TI third parties.
- **Language flexible**—programming languages supported include assembly, C, and C++.
- **Getting Started Quickly** help topics describe how to get started with Code Composer Studio.
- **Views and editors** section provides information various views that are available in Code Composer Studio.
- **Tasks** section provides topics help achieve specific tasks during project creation, build, debug and profiling.
- **References** section provides information additional information on Examples, Processor information and release notes.
- **Customer Support** section provides support information
- **Legal** section contains legal notices.

To re-display the Code Composer help again, select the **Help** → **Help Contents** command from the Help menu.

Where to Begin

An excellent source for getting started information may be found on Code Composer Studio v4 Quick Tips Wiki page . More detailed task oriented information about Code Composer Studio features may be found at Code Composer Studio v4 Wiki and Code Composer Studio V6 Wiki page.

Top of Page

DSP Knowledgebase

Search the latest FAQs, updates, bugs and troubleshooting online

News and Technical Support

For up to date information on Code Composer Studio and additional information on tasks such as how to create portable project please visit Code Composer Studio please go the Texas Instruments CCS v4 MediaWiki.

For technical support in using the Code Composer Studio, please browse through Wiki page and or CCS Forum E2E Forums

Top of Page

Release Notes

Read essential information about this release's defect fixes, work-arounds, and known software problems.

Submit Code Composer Documentation Feedback

1.1. Finding Your Way Around Code Composer Studio

Now that you have installed Code Composer Studio, you can proceed with product licensing and exploring the work environment. The following list is an introduction.

Creating a New CCS Project
Building Your Project
Common Debug Actions
Creating Custom Target Configuration Files
Connecting/Disconnecting a Target
Tracking Variables and Expressions

Submit Code Composer Documentation Feedback

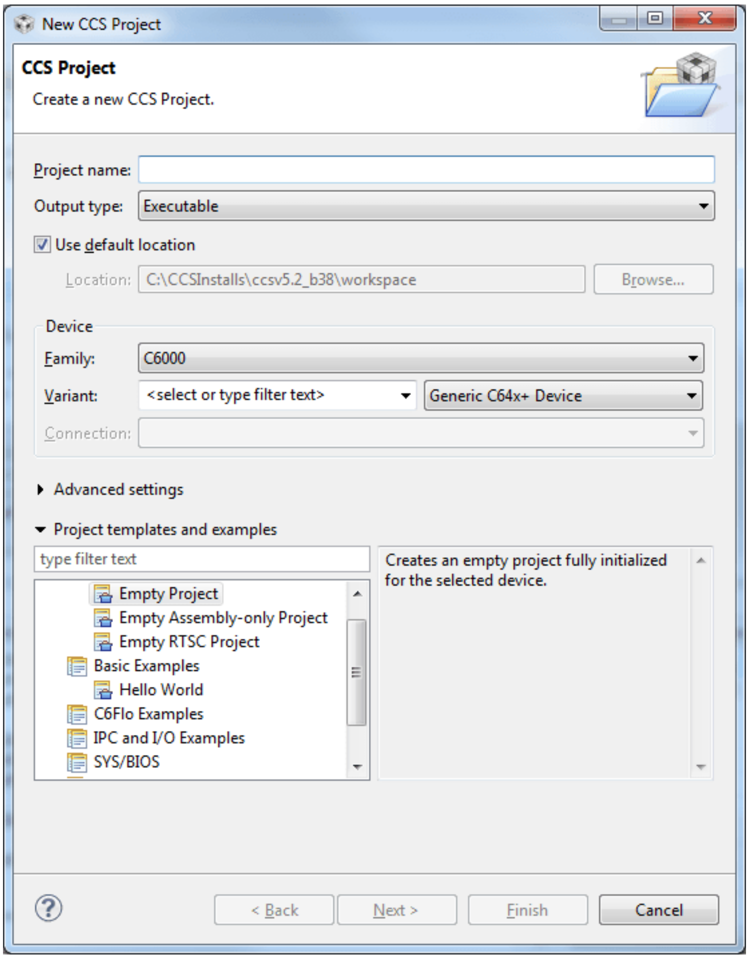
1.2. Creating a new CCS Project

This procedure helps you to create single or multiple new CCS Projects (multiple projects can be open simultaneously). Each project in the workspace must have a unique name.

1. Start Code Composer Studio from the desktop shortcut.

2. Select **Project > New CCS Project or File > New > CCS Project** action from the Workbench main menu.

3. When the **New CCS Project** wizard displays, type the **Project name**. You can optionally de-select the **Use default location** check box and set the project storage location. By default, the project will be created inside the workspace directory.
   Output Type allows you to toggle between creating an executable project or a library. You will then need to select Device Family and specific Device. If your specific device is not listed, then select one of the generic devices. Lastly, you may choose to select a project template/example as a starting point. You may also use TI Resource Explorer (Project > CCS Example Projects) to import additional example projects.
   Advanced settings allow you to override default compiler versions, executable format and linker.cmd files. If using a specific device, it is recommended to leave these settings as is.



New CCS Project

CCS Project

Create a new CCS Project.

Project name:

Output type: Executable

☑ Use default location

Location: C:\CCSInstalls\ccsv5.2_b38\workspace

Device

Family: C6000

Variant: <select or type filter text> Generic C64x+ Device

Connection:

Advanced settings

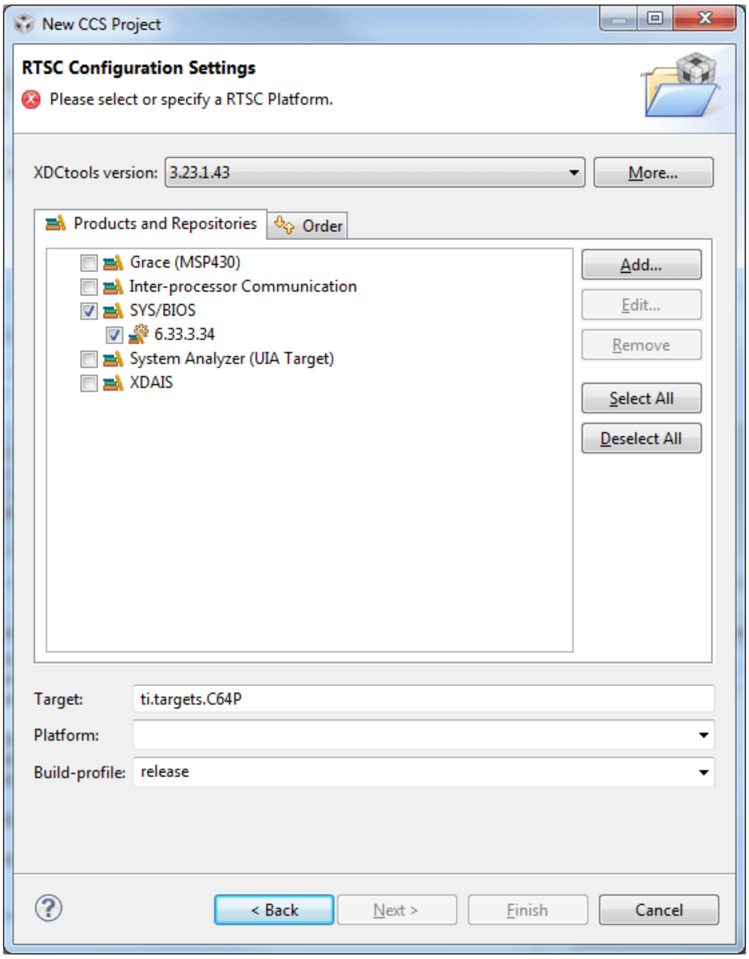
Project templates and examples

type filter text

- Empty Project
- Empty Assembly-only Project
- Empty RTSC Project
- Basic Examples
  - Hello World
- C6Flo Examples
- IPC and I/O Examples
- SYS/BIOS

Creates an empty project fully initialized for the selected device.

4. If you choose a Sys/Bios project you may need to fill in additional information such as platform. In this case "**Next**" button will be activated and you will be taken to next page in the wizard that allows you provide platform information and choose additional products for your project.



New CCS Project

RTSC Configuration Settings

☒ Please select or specify a RTSC Platform.

XDCtools version: 3.23.1.43

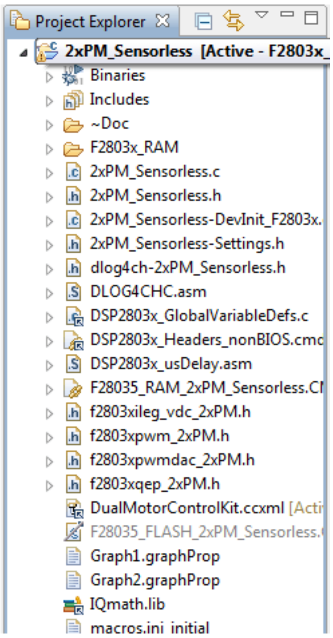
- □ Grace (MSP430)
- □ Inter-processor Communication
- ☑ SYS/BIOS
  - ☑ 6.33.3.34
- □ System Analyzer (UIA Target)
- □ XDAIS

Target: ti.targets.C64P

Platform:

Build-profile: release

The new project displays in the C/C++ Projects view and is set as the currently active project.



- 2xPM_Sensorless [Active - F2803x_
  - Binaries
  - Includes
  - ~Doc
  - F2803x_RAM
  - 2xPM_Sensorless.c
  - 2xPM_Sensorless.h
  - 2xPM_Sensorless-DevInit_F2803x.
  - 2xPM_Sensorless-Settings.h
  - dlog4ch-2xPM_Sensorless.h
  - DLOG4CHC.asm
  - DSP2803x_GlobalVariableDefs.c
  - DSP2803x_Headers_nonBIOS.cmc
  - DSP2803x_usDelay.asm
  - F28035_RAM_2xPM_Sensorless.CI
  - f2803xileg_vdc_2xPM.h
  - f2803xpwm_2xPM.h
  - f2803xpwmdac_2xPM.h
  - f2803xqep_2xPM.h
  - DualMotorControlKit.ccxml [Acti
  - F28035_FLASH_2xPM_Sensorless.(
  - Graph1.graphProp
  - Graph2.graphProp
  - IQmath.lib
  - macros.ini_initial

After creating the project, create your source files, set object libraries using the project Properties dialog, and add or link other files into the project.

7. To build your project, click the **Build Project** button on the main Workbench toolbar.

8. To launch a debug-session and download your application onto the target device, click the **Debug** button on the main Workbench toolbar (). Depending you device you may need to create a target configuration file (File ->New->Target Configuration) to specify target that your project should be loaded to.

Submit Code Composer Documentation Feedback

1.3. Building Your Project

Now that you have created a functional project, you can build it.

1. Select the project in Project Explorer and select **Project > Build Project**. You may also build all project in your workspace by selecting **Build All** menu action to build all the projects in your workspace.

2. The project should build successfully. The Console View displays the standard and error outputs of the build-tools. When the build is finished, the Problems View displays any errors or warnings.

3. After the project builds successfully, you can launch a debug-session and load the program onto the target. If you need to load the program manually, select **Run > Load Program** or Load Program icon on main toolbar. Code Composer creates a subdirectory for the currently active build-configuration (Debug or Release) within your project directory and stores the .out file in it.

Alternatively, you can launch a debug-session using the **Debug** toolbar button, and Code Composer will automatically perform an incremental build of your project before starting the debug-session. **Debug** is a single action that builds a project and launches the debugger using target configuration information contained in the project.

4. You will automatically be taken to the Debug perspective. To manually switch to the Debug perspective, select the Debug icon in the far right of the Code Composer window, or by selecting **Window** > **Open Perspective** > **CCS Debug**.

Note: Remember to reload the program by choosing **Run** > **Load Program** if you rebuild the project after making changes.

There are several options available for building projects:

- Full Builds — re-build and re-link all the source-files. Select **Project > Clean and then Project -> Build to re-build the currently selected project, or Project > Rebuild All to re-build all the open projects in the workspace.**
- Incremental Builds — build and re-link only the source-files that were modified since last build. Select **Project > Build Project** to build the currently active project, or **Project > Build All** to build all the open projects in the workspace.
- Build Automatically — performs an incremental build automatically whenever any source-files or relevant header-files are saved.

Submit Code Composer Documentation Feedback

1.4. Common Debug Actions

Start debugging

Image	Name	Description	Availability
	New Target Configuration	Creates a new target configartion file.	File New Menu Target Menu
	Debug	Opens a dialog to modify existing debug configurations. Its drop down can be used to access other launching options.	Debug Toolbar Target Menu
	Connect Target	Connect to hardware targets.	TI Debug Toolbar Target Menu Debug View Context Menu
	Terminate All	Terminates all active debug sessions.	Target Menu Debug View Toolbar

Program execution

Image	Name	Description	Availability
	Halt	Halts the selected target. The rest of the debug views will update automatically with most recent target data.	Target Menu Debug View Toolbar
	Run	Resumes the execution of the currently loaded program from the current PC location. Execution continues until a breakpoint is encountered.	Target Menu Debug View Toolbar
	Run to Line	Resumes the execution of the currently loaded program from the current PC location. Execution continues until the specific source/assembly line is reached.	Target Menu Disassembly Context Menu Source Editor Context Menu
	Go to Main	Runs the programs until the beginning of function main in reached.	Debug View Toolbar
	Step Into	Steps into the highlighted statement.	Target Menu Debug View Toolbar
	Step Over	Steps over the highlighted statement. Execution will continue at the next line either in the same method or (if you are at the end of a method) it will continue in the method from which the current method was called. The cursor jumps to the declaration of the method and selects this line.	Target Menu Debug View Toolbar
	Step Return	Steps out of the current method.	Target Menu Debug View Toolbar
	Reset	Resets the selected target. The drop-down menu has various advanced reset options, depending on the selected device.	Target Menu Debug View Toolbar
	Restart	Restores the PC to the entry point for the currently loaded program. If the debugger option "Run to main on target load or restart" is set the target will run to the specified symbol, otherwise the execution state of the target is not changed.	Target Menu Debug View Toolbar
	Assembly Step Into	The debugger executes the next assembly instruction, whether source is available or not.	TI Explicit Stepping Toolbar Target Advanced Menu
	Assembly Step Over	The debugger steps over a single assembly instruction. If the instruction is an assembly subroutine, the debugger executes the assembly subroutine and then halts after the assembly function returns.	TI Explicit Stepping Toolbar Target Advanced Menu

Program and symbol download

Image	Name	Description	Availability
	Load Program	Loads the COFF file (`*.out`) produced by building your program onto the actual or simulated target board prior to execution.	Target Menu Debug Toolbar Modules View Toolbar
	Load Symbols	Loads symbol information only. It is useful to load only symbol information when working in a debugging environment where the debugger cannot or need not load the object code, such as when the code is in ROM. When you load symbols, you are first clearing the existing symbol table before you add new symbols.	Target Menu Modules View Toolbar
	Reload Program	Reloads the COFF file (`*.out`) onto the actual or simulated target board.	Target Menu Debug Toolbar Modules View Toolbar
	Add Symbols	Adds symbols by appending symbol information to the existing symbol table.	Target Menu Modules View Toolbar
	Edit Source Lookup	Specifies the locations of the source files for source level debugging.	Launch Configuration Dialog Tab Debug View Context Menu

See Also:

- Working with the Debug View

- Working with the Modules View
- Starting a debug session
- Specifying the target configuration

Submit Code Composer Documentation Feedback

1.5. Setting Target Configuration

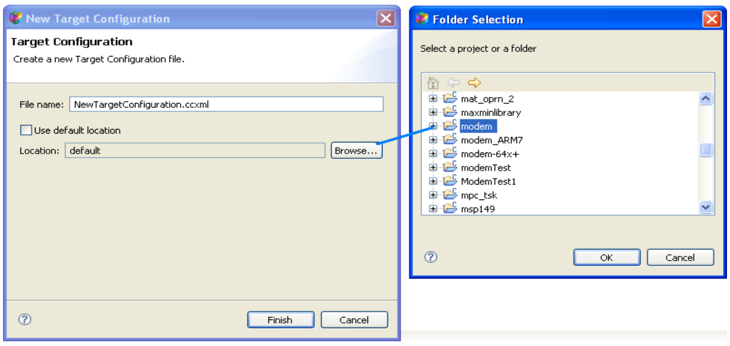
In migrating from the existing Code Composer Studio, the target configuration selection method and organization have changed. These changes include:

1. Target configuration now uses a flexible XML schema that helps Texas Instruments and developers define the specific details. Instead of using the registry, Code Composer uses a hierarchy of related XML files to define a connection and board/device combination. Because the target configurations are XML files, they are easily deleted, copied, and displayed.

2. Separating the connection technique (XDS560, simulation, emulation) from the device/board selected. This implies:
   - You can change the emulation type a target configuration uses.
   - Third-party vendors do not need to create configurations that are device/board dependent. Every user chooses the connection type and the needed board.

3. The target setup is now integrated into the Code Composer Studio Integrated Development Environment. Users can construct target configuration files directly in Code Composer Studio. Select File->New -> Target Configuration to create new target configuration file, when you specify the name and location for the file, the setup editor opens, allowing you to select the debug target.

4. Target configuration files can be added to individual projects. These projects will automatically use the target configuration from the project, when debugger is started using Launch Debugger action. If you create a new CCS project and select a specific device, then in most cases a target configuration file will be automatically created for you with default emulator. You may adjust default connection through project properties (Right click on a project and select Properties). You may also open .ccxml file in your project to inspect connection and device settings.



New Target Configuration

Target Configuration

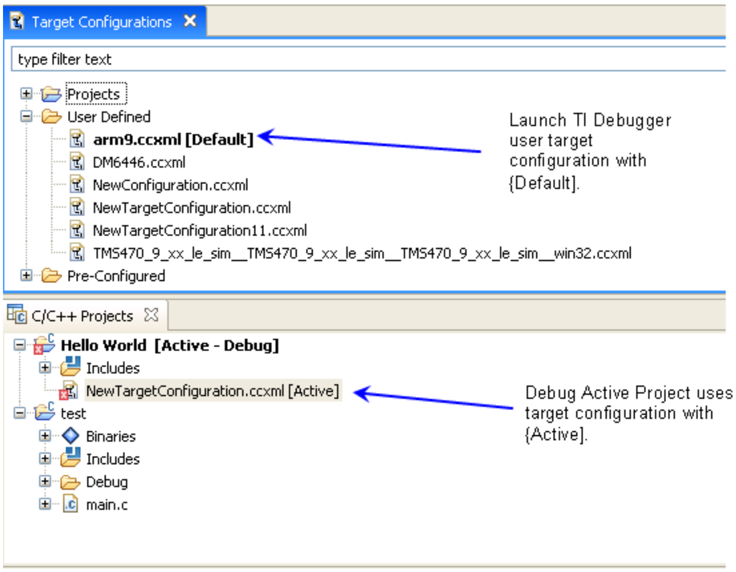
Create a new Target Configuration file.

File name: NewTargetConfiguration.ccxml

□ Use default location

Location: default

5. Target configuration files are *automatically* added to user specific folder. The default location for this folder is "Documents and Settings\userid\ti\CCSTargetConfigurations". This location may be adjusted through Windows->Preferences. A configuration can be designated as the `[Default]` configuration. Default configuration will be used by Debug command.



Target Configurations
type filter text
Projects
User Defined
arm9.ccxml [Default]
DM6446.ccxml
NewConfiguration.ccxml
NewTargetConfiguration.ccxml
NewTargetConfiguration11.ccxml
TMS470_9_xx_le_sim__TMS470_9_xx_le_sim__TMS470_9_xx_le_sim__win32.ccxml
Pre-Configured
C/C++ Projects
Hello World [Active - Debug]
Includes
NewTargetConfiguration.ccxml [Active]
test
Binaries
Includes
Debug
main.c

Launch TI Debugger user target configuration with [Default].

Debug Active Project uses target configuration with [Active].

6. The **Debug** command uses the **`[Active]`** project configuration or falls back on the **`[Default]`** Configuration. There may be multiple target configuration files in a project, but only one is designated **`[Active]`**.

Submit Code Composer Documentation Feedback

1.6. Target Connect/Disconnect

Code Composer Studio makes it easy to dynamically connect and disconnect hardware targets by using the Connect/Disconnect functionality. Connect/Disconnect helps you to disconnect from your hardware target and optionally restore the previous debug state (e.g. breakpoints) when reconnecting.

Connect/Disconnect functioanlity is applicable to hardware targets only.

Note: When the **Debug Project** command performs, the hardware target is automatically connected.

Launching debugger from target configurations view does not automtically connect to the target. Connection to the target can be established by clicking on connect button.

After connecting to the target (except for the initial connection), use the **Run > Advanced> Restore Debug State** command or Restore Debug State toolbar button Selecting this option enables every breakpoint that was disabled at disconnect. You can also reset them by selecting the **Toggle Breakpoints** command from the context-sensitive menu.

Submit Code Composer Documentation Feedback

1.7. Overview

Introduction

CCS Graphs are used to plot data in target memory visually or do some processing on the data and display as graphs. This works on all ISAs and simulator and Hardware platforms.

Data can be plotted by

- Manual Refresh
- Breakpoint and Refresh Breakpoint action can be set to
  - **Update View**
  - **Refresh All Windows**
- Target Halt and Refresh
- Continuous Refresh

Graph Types

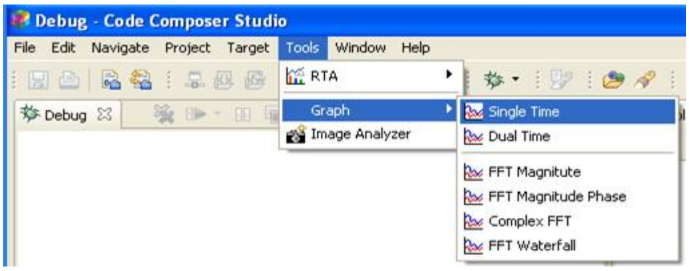
The following type of graphs is available to the user:

- Single Time
- Dual Time
- FFT Magnitude
- Complex FFT
- FFT Magnitude & Phase
- FFT Waterfall

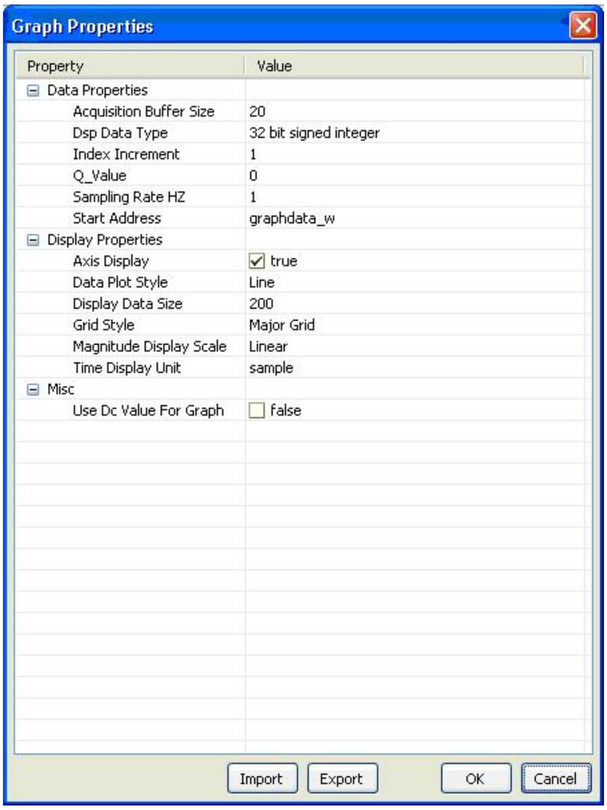
Accessing Graph Analysis Tools

Preconditions: A Target must be connected

- From the main menu click on ‘Tools->Graph’
- Then Click on a specific graph type from the graph submenu



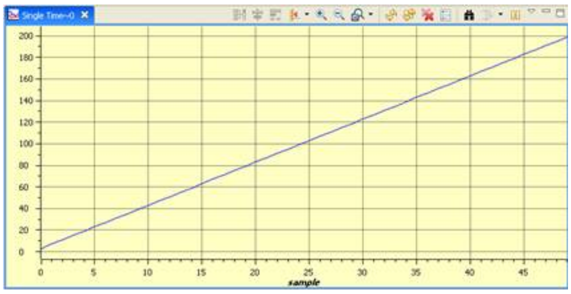
- The user is presented with a graph properties dialog


Graph Properties

Property	Value
⊟ Data Properties	
Acquisition Buffer Size	20
Dsp Data Type	32 bit signed integer
Index Increment	1
Q_Value	0
Sampling Rate HZ	1
Start Address	graphdata_w
⊟ Display Properties	
Axis Display	☑ true
Data Plot Style	Line
Display Data Size	200
Grid Style	Major Grid
Magnitude Display Scale	Linear
Time Display Unit	sample
⊟ Misc	
Use Dc Value For Graph	□ false

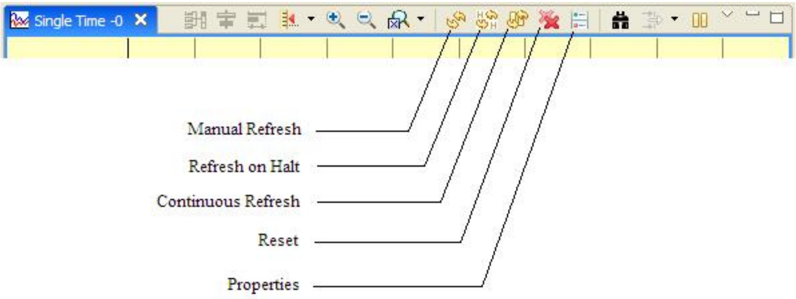
Specify the graph properties and click OK. Graph properties can also be saved by clicking on the Export button. User will need to specify a file name with default extension .graphProp. Later the set of properties can be restored by clicking on the Import button and selecting the desired file in which the properties were saved

The graph(s) is/are drawn in one or more views



Graph-Specific Toolbar Buttons

- **Refresh**: Fetches and displays new data from the target
- **Refresh On halt**: Sets graph behavior on target halt. Enables/Disables graph fetching and displaying data on encoutering a target halt. By default it is Enabled
- **Continuous Refresh**: Causes the view to refresh periodically
- **Reset**: Erases the graph plot
- **Properties**: Show the properties dialog



2.1. Breakpoint Overview

A breakpoint will trigger it's action when the specified condition is evaulated to true. A program address breakpoint, the condition is the location (program address) and the action is to suspend the program execution. Below is a list of breakpoint types that you can set for your debug session, these options are dependent on the target that you are connected to:

- **A program address breakpoint trigger its action when the execution of a program reaches the location where the breakpoint is set.**
- **A watchpoint trigger its action when the variable or data address is accessed.**
- **A chained breakpoint trigger its action when both conditions are evaluated to true.**
- **A event counter trigger its action when the count value reaches the specified condition.**
- **A stack overflow breakpoint trigger its action when the specified stack overflow.**
- **And others...**

Breakpoints can be set on a line of source code in the **Source Editor** or a disassembled instruction in the **Disassembly view**, on a method in the Outline or Project Explorer view, or in the **Breakpoints view**. After a breakpoint is set, it can be enabled or disabled. It can also have actions or conditions associated with it.

Breakpoints can be enabled and disabled via the context menu or throught the checkboxes in the **Breakpoints view**.

- An enabled breakpoint causes a thread to suspend whenever the breakpoint is encountered. Enabled breakpoints are drawn with a blue circle [●] and have a checkmark overlay once successfully installed. If you add a hardware breakpoint, the icon displays like this [✦]. Note that Code Composer automatically uses software breakpoint when you toggle a breakpoint in the **Source Editor** or **Disassembly view**. If software breakpoint can not be set on the target due to readonly memory, it creates hardware breakpoint instead.
- A disabled breakpoint does not cause threads to suspend. Disabled breakpoints are drawn with a gray circle [●].

Breakpoints stop the execution of the program. While the program is stopped, you can examine the state of the program, examine or modify variables, examine the call stack, etc. The **Breakpoints view** lets you create or control one or more breakpoints.

Breakpoint Properties

You can view breakpoint properties within the **Properties** dialog. If you have selected a breakpoint in the Breakpoints view you can right click and select *Properties...* from the context menu to open the **Properties** dialog. The breakpoint properties are saved in the workspace, so you do not need to reset them in between debug sessions. You can only modify breakpoint property when the breakpoint is installed onto the target.

Software vs. Hardware Breakpoints

A software breakpoint is implemented as an opcode replacement, whereas a hardware breakpoint is implemented internally by the hardware. There is no limit to the number of software breakpoints you can set. However, because hardware breakpoints require on-chip analysis resources, you are limited to the available resources for your target.

Intrusive vs. Nonintrusive Breakpoints

An intrusive breakpoint halts the target. Breakpoints on simulators are always intrusive. A nonintrusive breakpoint won't halt the target when evaluating a condition.

See Also:

- Breakpoints View
- Adding Breakpoint
- Modifying Breakpoint Properties

Submit Code Composer Documentation Feedback

2.1.2. Breakpoints View

The Breakpoints View lists all the breakpoints you have set in the workbench.

You can double-click a breakpoint to display its location in the editor (if applicable). You can also enable or disable breakpoints, delete them, add new ones, and modify it's properties.

The columns in the Breakpoints View provide a summary for each breakpoint, it is automatic populated once the target is connected.

- **Location the location of the breakpoint, not all breakpoint will have a location. i.e Event, Counter, etc...**
- **Name the name of the breakpoint**
- **Condition the condition that determines when the breakpoint is trigger**
- **Count the hit count of the breakpoint**
- **Action the action to perform when the breakpoint is triggered**

The following is a list of available actions in this view:

Image	Name	Description	Availability
	Assign to Target	Assign a breakpoint to a particular target, as selected in the Targets dialog box that opens.	Context menu
⊟	Collapse All	Collapses all of the items in the view.	View action
	Deselect Default Working Set	De-select the default working set.	View action
☐	Disable	Changes the selected breakpoint(s) to be disabled.	Context menu
☑	Enable	Changes the selected breakpoint(s) to be enabled.	Context menu
⊞	Expand All	Expands all of the items in the view.	View action
	Export Breakpoints...	Opens the export breakpoints wizard.	Context menu
	Go to File	Opens the corresponding location of the breakpoint in the editor.	Context menu and view action
	Group By...	Select an alternate grouping for your breakpoints or create your own.	View action
	Import Breakpoints...	Opens the import breakpoints wizard.	Context menu
	New	Adds a new breakpoint, additional breakpoint types are available in the drop down once the target is connected.	View action
	Properties...	Opens the breakpoint properties dialog.	Context menu
✖	Remove All	Removes all breakpoints from the view.	Context menu and view action
✖	Remove Selected Breakpoints	Removes only the selected breakpoint(s) from the view.	Context menu and view action
	Resolve Path	Resolves breakpoint source location base on the module setting in your source search path. If you are not able to enable the breakpoint and the breakpoint is a source line breakpoint, than you can use this option to *try* to resolve the breakpoint with the current loaded symbol. The breakpoint icon has an overlay **warning** icon indicates that the breakpoint's source location is unresolve and you can use this action to resolve the breakpoint's source location.	Context menu
	Select All	Selects all of the breakpoints in the view.	Context menu
	Select Default Working Set	Choose which working set is the default one.	View action
	Show Full Paths	Sets the location column in the breakpoints view to show fullpath.	View action
	Skip All	Sets all breakpoints to be skipped.	View action
	Working Sets...	Opens the working sets dialog.	View action

See Also:

- Breakpoint Overview
- Breakpoint Icons
- Adding Breakpoint
- Modifying Breakpoint Properties

Submit Code Composer Documentation Feedback

2.1.3. Breakpoint Icon Indicators

A breakpoint icon indicates the type of breakpoint you have set on your target, whether it is a software breakpoint, a hardware breakpoint, a watchpoint, a counter, etc... As well as the conditions and actions the breakpoint has.

The following is a list of some of the breakpoint icon:

Icon	Description
⏱	Counter
◈	Hardware Breakpoint
◇	(Fuzzy) Hardware Breakpoint for Non-Active Debug Context Applies to source editor only.
●	Software Breakpoint
◎	(Fuzzy) Software Breakpoint for Non-Active Debug Context Applies to source editor only.
	Watchpoint
	Watchpoint with Read Condition
✎	Watchpoint with Write Condition
»»	Breakpoint Hit Overlay Occure when the breakpoint is triggered and re-run by the debugger due to condition is not satisified

⋙	Breakpoint Trigger Overlay Occure when the breakpoint is triggered, condition is satisified
✓	Installed Breakpoint Overlay Breakpoint is installed by the debugger onto the target.
▶	Non-Default Action Overlay Breakpoint has an action other than halting the target.
⚠	Source Unresolved Warning Overlay Breakpoint with source unresolved, required user action to resolve source location, see Resolve Path Action

See Also:

- Breakpoint Overview
- Breakpoints View
- Adding Breakpoint
- Modifying Breakpoint Properties

Submit Code Composer Documentation Feedback

2.1.4. Adding Breakpoint

Breakpoints can be add in varies different ways.

- **Double clicking in the Source Editor annotation margin (vertical ruler) or the Disassembly view annotation margin to toggle a program address breakpoint. A new breakpoint will appears on the annotation margin and if there is no valid assembly instruction associated with the toggled line, than the breakpoint will be automatic shifted to the next valid address.**
- **Right clicking in the Source Editor annotation margin or the Disassembly view annotation margin to show the avaliable options to set a program address breakpoint.**
- **Highlight a variable (text) in the Source Editor source and right click on the highlighted text, a list of avaliable new breakpoint options will be populated in the *New Breakpoint* sub-menu.**

  Selecting one of these options will either add a new breakpoint to the target if there is no additional input is required, otherwise a dialog will prompt for additional inputs. The list of option is only avaliable when the target is connected and the number of option is target dependent.
- **Right clicking on the method name in the Project Explorer or the Outline view to show the available options to set a program address breakpoint for the selected method.**
- **Within the Breakpoint view, pressing the *new* toolbar button will prompt to create a software breakpoint. And selecting the down arrow beside the *new* toolbar button will show the aviable breakpoint options based on the selected target in the Debug view.**

See Also:

- Breakpoint Overview
- Breakpoints View
- Modifying Breakpoint Properties

Submit Code Composer Documentation Feedback

2.1.5. Modifying Breakpoint Properties

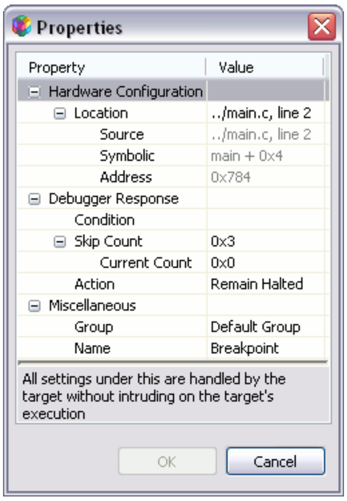
You can modify the properties of a breakpoint in the **Breakpoints view** by clicking in the column cell or throught the Properties dialog by right click on a breakpoint and select *Properties...*. The context menu action for properties is only avaliable when the breakpoint is installed onto the target.

Editing Properties in The Breakpoint View

Property value is editable when the the breakpoint is installed on the target, simply clicking in the cell to begin editing the value. If changing a value required additional input, than a dialog will prompt for additional input.

Editing Properties in The Properties Dialog

Breakpoint properties are dynamic generated by the debugger, the property values in the breakpoint property dialog are depended on the analysis unit that you have on your target. Changing the value of a property can cause other property to be added or removed from the property tree.



Property	Value
⊟ Hardware Configuration	
⊟ Location	../main.c, line 2
Source	../main.c, line 2
Symbolic	main + 0x4
Address	0x784
⊟ Debugger Response	
Condition	
⊟ Skip Count	0x3
Current Count	0x0
Action	Remain Halted
⊟ Miscellaneous	
Group	Default Group
Name	Breakpoint

All settings under this are handled by the target without intruding on the target's execution

Property value can have the following type, a string, an integer, and a boolean. For free form text input, an expression can be use as an input and the result of the expression evalution will be placed in the free form text box.

- **A string property can be either presented in free form text box or a combo-box.**
- **An integer property can be either presented in a free form text box or a comb-box.**
- **A boolean property is presented with a check-box.**

Property value that is not editable has it's text grayed out, otherwise you can click on the cell in the *Value* column to edit the value, or press F2 on your keyboard. When done, press the *OK* button, the changes will not be submit to the debugger if *Cancel* button is selected.

See Also:

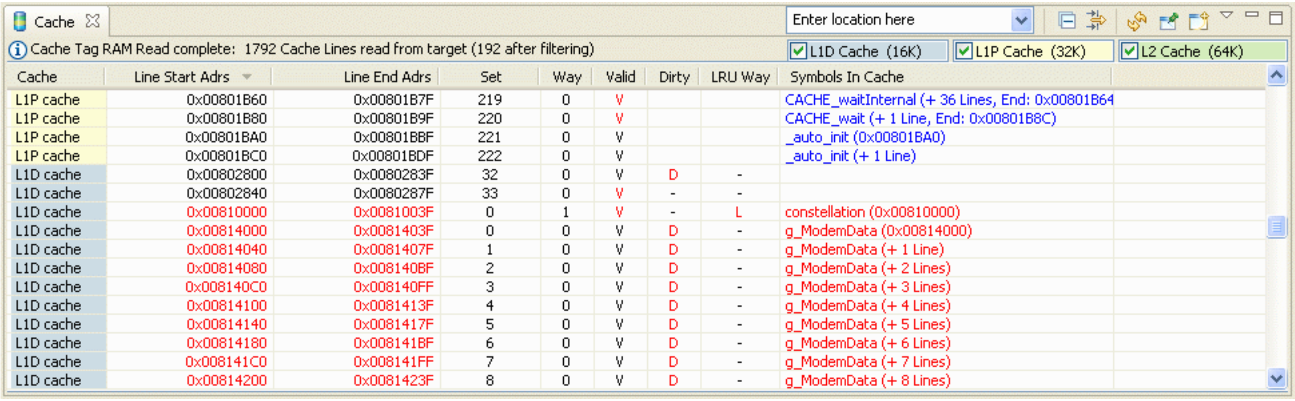
- Breakpoint Overview

- Breakpoints View
- Adding Breakpoint

Submit Code Composer Documentation Feedback

2.2. Cache View

The Cache tag viewer displays the contents of cache in tabular form, providing a 'map' that shows what memory locations are being cached and their associated cache tag RAM status bits. The contents include: tag address, set , way, valid bit, dirty bit, Least Recently Used (LRU) bit. The Cache Tag RAM Viewer provides a static view into cache that lets programmers verify their optimization methods. They can (for example) check whether what is expected to be in cache is in fact in cache. They can also fine tune memory layouts and access patterns. For instance the LRU status can be actively managed to retain certain data structures in cache.



Cache Tag RAM Read complete: 1792 Cache Lines read from target (192 after filtering)

☑ L1D Cache (16K) ☑ L1P Cache (32K) ☑ L2 Cache (64K)

Cache	Line Start Adrs	Line End Adrs	Set	Way	Valid	Dirty	LRU Way	Symbols In Cache
L1P cache	0x00801B60	0x00801B7F	219	0	V			CACHE_waitInternal (+ 36 Lines, End: 0x00801B64
L1P cache	0x00801B80	0x00801B9F	220	0	V			CACHE_wait (+ 1 Line, End: 0x00801B8C)
L1P cache	0x00801BA0	0x00801BBF	221	0	V			_auto_init (0x00801BA0)
L1P cache	0x00801BC0	0x00801BDF	222	0	V			_auto_init (+ 1 Line)
L1D cache	0x00802800	0x0080283F	32	0	V	D	-	
L1D cache	0x00802840	0x0080287F	33	0	V	-	-	
L1D cache	0x00810000	0x0081003F	0	1	V	-	L	constellation (0x00810000)
L1D cache	0x00814000	0x0081403F	0	0	V	D	-	g_ModemData (0x00814000)
L1D cache	0x00814040	0x0081407F	1	0	V	D	-	g_ModemData (+ 1 Line)
L1D cache	0x00814080	0x008140BF	2	0	V	D	-	g_ModemData (+ 2 Lines)
L1D cache	0x008140C0	0x008140FF	3	0	V	D	-	g_ModemData (+ 3 Lines)
L1D cache	0x00814100	0x0081413F	4	0	V	D	-	g_ModemData (+ 4 Lines)
L1D cache	0x00814140	0x0081417F	5	0	V	D	-	g_ModemData (+ 5 Lines)
L1D cache	0x00814180	0x008141BF	6	0	V	D	-	g_ModemData (+ 6 Lines)
L1D cache	0x008141C0	0x008141FF	7	0	V	D	-	g_ModemData (+ 7 Lines)
L1D cache	0x00814200	0x0081423F	8	0	V	D	-	g_ModemData (+ 8 Lines)

The Cache View will remain disabled until a TMS320C64X+ CPU (either a simulator or a connected emulation target) is selected in the CCS Debug View.

Devices that do not support Cache Tag RAM Visibility

Cache visibility is only supported on TMS320C64X+ devices. TMS320C64X+ devices based on the GEM 1.0,1.1 and 2.0 silicon revisions do not support the cache tag RAM interface that the Cache View uses to access cache information with. These devices include:

- **TCI6482 (Himalaya 1.x, 2.0) silicon**
- **DM6455 (DaVinci 1.x) silicon**
- **IVA21, IVA22 silicon**

Note: Drivers for development boards that provide USB-based emulator support often *do not support cache visibility*. Using the JTAG header with a JTAG emulator on such boards may be required in order to enable the Cache View.

What is Cache Tag RAM?
The cache tag RAM stores cache states. This includes:

- **Set: A collection of line-frames in a cache that a single address can potentially reside in. A direct-mapped cache (e.g. L1P) contains one line-frame per set, and an N-way set-associative cache contains N line-frames per set. A fully-associative cache has only one set that contains all of the line-frames in the cache.**
- **Way: In a set-associative cache, each set in the cache contains multiple line-frames. The number of line-frames in each set is referred to as the number of ways in the cache. The collection of corresponding line-frames across all sets in the cache is called a way in the cache. For instance, L2 cache is a 4-way set-associative cache that has 4 ways, and each set in the cache has 4 line-frames associated with it, one associated with each of the 4 ways. As a result, any given cacheable address in the memory map has 4 possible locations it can map to in a 4-way set-associative cache.**
- **Tag address: address bits that identify the physical line of memory that the cache line is assoiated with**
- **Valid: flag indicatin whether the line is in cache (true) or not (false)**
- **Dirty: flag that, if true, indicates that the line was written to by the CPU but has not yet been written to the associated mapped memory**
- **LRU: flag indicating that the cache line is the least recently used cache line in the cache set.**

The following is a list of available actions in this view:

Image	Name	Description	Availability
Enter location here ▾	Search for Address	Searches for the specified address in the displayed cache lines and, if a cache line that contains the address is found, scrolls the table to display it and selects the row in the table. Addresses can be specified as hex values (e.g. 0x12345678), symbol names (e.g. main or &myDataStructure), or a previously entered address can be selected from the edit box's drop-down list.	Toolbar Action
⊞	Show Cache Line Details	Configure the Cache View to show details about each cache line, with each row in the table displaying details about a single cache line. See the "Show each symbol in a separate row" context menu option for additional configuration details. This icon is only visible when the display is in "Cache Line Summary" mode (see below).	Toolbar Action
⊟	Show Cache Line Summary	Configure the Cache View to show a summary of the lines in cache, with each row in the table summarizing all conseutive lines from the same cache. This icon is only visible when the display is in "Cache Line Details" mode (see above).	Toolbar Action
	Filter...	Filters the lines of cache that are displayed in the view based on user-specified cache line address range, cache levels of interest, and cache line attributes.	Toolbar Action
	Refresh	Refreshes the view	Toolbar Action
	Pin	Once pinned, the view will only show the cache lines for the context it was pinned to instead of displaying the cache lines for the target selected in the Debug View.	Toolbar Action
	New Cache View	Opens a new cache view.	Toolbar Action
	Edit Cache Line	Opens a memory view at the address of the selected cache line. Enables the Memory View's Memory Analysis feature to highlight addresses with memory differences or cache attributes of interest. (Select "configure" from the Memory vie'ws context menu to specify what is highlighted.)	Context Menu
	Copy selected cell	Copies the text of the selected cell to the clipboard)	Context Menu
	Show Dashboard	Controls whether the toolbar used to display the cache level checkboxes and cache size info is displayed or not .	Context Menu

≠	Highlight Memory Differences	When checked, cache lines that contain one or more addresses that have different values in cache than in mapped memory are displayed in bold text. Note that memory difference analysis slows down the responsiveness of the cache view..	Context Menu
a/z	Show each symbol in a separate row	When checked and the "Show Cache Line Details" display option is active, each symbol that is resident in the cache is displayed on a separate row in the Cache View display table. If there are multiple symbols resident in a cache line, the same cache line information will be repeated for each symbol in that cache line.	Context Menu
▽	Auto Update	When checked (default), the Cache View is updated automatically whenever the target halts. When unchecked, the user must click the Refresh icon in order to update the Cache View display.	Toolbar Menu Item
▽	Save As Xml...	Saves the displayed cache lines in a user-specified file in XML format.	Toolbar Menu Item

Using the Cache View

In order to maximize the cache performance of your software, you need to make sure your key functions and data structures are resident in cache and not being bumped from cache (either entirely or partially). To determine what is resident in cache at any given point in your program, halt the processor and check to make sure that the address ranges or symbols associated with these functions and data structures are displayed by the Cache View. If they are only partially displayed (or if the cache set that was used to hold them previously now contains different data), there may be conflicting requests for caches that are impacting the performance of your software . You can use the Cache View to determine what other lines of memory they have been bumped by, which data structures are competing for cache lines, how data structures are aligned, whether there is any 'false sharing' of cache lines between two different applications, etc. Tip: ***Double clicking on a row in the table will toggle the view between the "Cache Line Summary" mode and "Cache Line Details" mode. Use the Cache Line Summary mode to quickly scroll to an address range of interest, and then double click on a row of interest to see the details about individual cache lines.***

Tooltips

Tooltips are used extensively in the user interface to provide information about the item that the mouse is hovering over.

Sorting Options

Clicking on the top cell of a column in the Cache View table sorts the cache lines by the contents of that column. Clicking on the same column header again reverses the sort order.

- **To view cache information in top-down fashion (first by cache type, then by address), sort by the Cache column.**
- **To analyze whether a particular address is cached, sort by Start Address.**
- **To group all the dirty lines together, sort by the Dirty column.**
- **To see which cache lines are at risk of being evicted, sort by the LRU (Least Recently Used) column.**
- **To analyze the layout of one kind of cache (such as assessing conflicting lines), sort by Set**
- **To determine an optimal data access pattern in order to optimize cache performance, sort by Way.**
- **Tip:** ***You can open 2 Cache Views side-by-side with one showing way 0 and the other showing way 1 of L1D cache.***

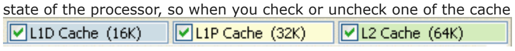
There are two main types of sorting strategies:

- ***Address priority*: sorting by Cache, Start Address, End address**
- ***Cache Layout*: sorting by Set, Way, LRU (least recently used way).**

The cache layout sorting strategy is more suitable for cache performance optimization tasks, whereas address sorting is more suitable for debugging multi-core coherence scenarios. For details on how to optimize your application's cache performance, please refer to the TMS320C64x+ DSP Cache User's Guide (SPRU862).

Cache Dashboard

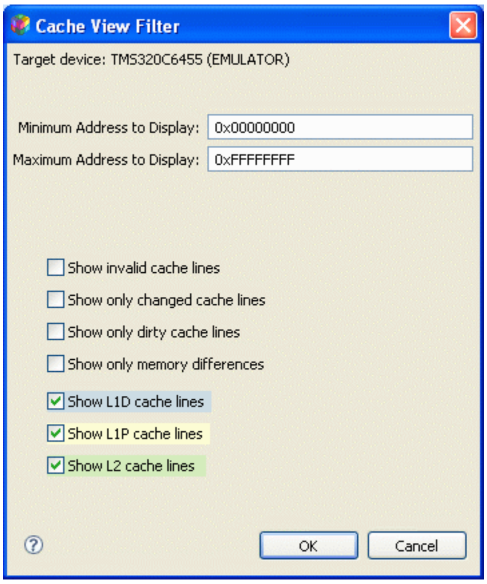
The Dashboard shows the levels of cache supported by the CPU that is in focus for the Cache View, the background coloring used to identify that cache level in the display, the amount of cache that the cache configuration registers are currently configured for, and a checkbox to control whether that level of cache is displayed in the Cache View or not. The CacheView does not alter the state of the processor, so when you check or uncheck one of the cache level checkboxes in the dashboard, you are configuring what is displayed, not what the target processor configuration is.



☑ L1D Cache (16K) ☑ L1P Cache (32K) ☑ L2 Cache (64K)

Filtering Options

In order to make it easier to see cache lines of interest, you can specify filtering options that control which cache lines are displayed in the Cache View. Clicking on the **Filter** icon opens up the Cache View Filter dialog box:



Cache View Filter

Target device: TMS320C6455 (EMULATOR)

Minimum Address to Display: 0x00000000

Maximum Address to Display: 0xFFFFFFFF

☐ Show invalid cache lines

☐ Show only changed cache lines

☐ Show only dirty cache lines

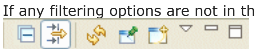
☐ Show only memory differences

☑ Show L1D cache lines

☑ Show L1P cache lines

☑ Show L2 cache lines

If any filtering options are not in their default settings (as shown in the above image), the Filter icon is highlighted ('pushed in').



This provides a quick way to determine if you are seeing all of the cache lines that are in cache or if some are being filtered out.

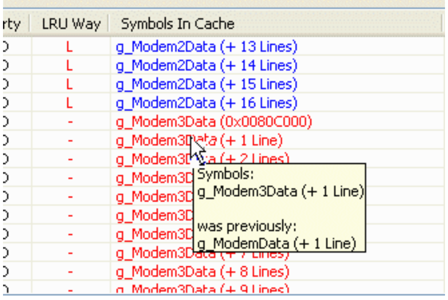
The **Minimum Address** and **Maximum Address** entries allow you to configure the Cache View to display only cache lines that reside in that range of addresses. You can enter addresses either as hex values (e.g. 0x12345678), using symbol names (e.g. main, &myDataStructure).

Restricting the range of addresses that are displayed in the Cache View can be useful, for example, when you are working on a multicore device and are debugging multicore cache coherency issues involving shared memory. You can open two cache views side by side, pin one of them to one CPU core and the other to another CPU core and configure both Cache Views to display only the cache lines in the address range used by shared memory.

The **Cache Level checkboxes** allow you to control whether that cache level is displayed or not. By default, all cache levels are displayed.
Note that these Cache Level checkboxes are funtionally equivalent to the cache level checkboxes displayed in the **Dashboard**. They are provided in the Filter Dialog so that you can hide the dashboard (via the context menu) if you wish to use the Cache View in a narrow window layout.

The **Show Only Changed Cache Lines** option filters out all ways and sets that have not changed since the last time the CPU halted. Cache line attributes that have changed since the last time the target halted are displayed in red text.

With the Cache View configured to display cache line details, tooltips provide information that allows you to determine which cache lines have been evicted. Hovering the mouse over the Start Address of a changed cache line will display the previous cache line start address that was used for that way & set the last time the target was halted. Hovering the mouse over the Symbols of a changed cache line will generate a tooltip that identifies the previously cached symbols, allowing you to see what symbols were bumped from cache.



rty	LRU Way	Symbols In Cache
)	L	g_Modem2Data (+ 13 Lines)
)	L	g_Modem2Data (+ 14 Lines)
)	L	g_Modem2Data (+ 15 Lines)
)	L	g_Modem2Data (+ 16 Lines)
)	-	g_Modem3Data (0x0080C000)
)	-	g_Modem3Data (+ 1 Line)
)	-	g_Modem3D a (+ 2 Lines)
)	-	g_Modem3D
)	-	g_Modem3D
)	-	g_Modem3D
)	-	g_Modem3D
)	-	g_Modem3Data (+ 7 Lines)
)	-	g_Modem3Data (+ 8 Lines)
)	-	g_Modem3Data (+ 9 Lines)

Symbols:
g_Modem3Data (+ 1 Line)

was previously:
g_ModemData (+ 1 Line)

See Also:

- [SPRU862 - TMS320C64x+ DSP Cache User's Guide]
- [CCS4 MediaWiki topic: Cache Visualization]
- [SPRU871�TMS320C64X+ Megamodule Peripherals Reference Guide]

2.3.1. Debugger Options Overview

Debugger Options in the Control Panel view allows users to adjust debugger behaviour during a debug session. Options adjusted in this view take immediate effect and are applicable only to CPU selected in Debug View. Debugger option changes will not persist between debugger launches unless user selects "Remember My Settings".

Note that the following debug options can be accessed at two different levels. They can either be accessed from the launch configuration (click on Debug button > select Debug... > select your launch configuration > select the Target tab) or they can be accessed from the control panel view (when a debug session has been started, click on Tools > Debugger Options).

The debugger options in the launch configuration can be used to modify the options before a debug session has started, whereas the debugger options in the control panel view can be used to modify the options for the currently running (or more accurately, the currently selected) debug session.

When options are modified in the Control Panel view, the changes will take effect right away (i.e. there is no OK or Cancel button). However, when options are modified in the launch configuration, the changes will take effect next time the configuration is launched.

The developer can choose to save the options selected in the Control Panel view back into the launch configuration by clicking on "Remember My Settings" (see below for more details).

The following is an explanation of all the Debugger Options:

Program Load Options

- **Disable all breakpoints when loading a different program:**
  When enabled, all breakpoints will be disabled when a different program is loaded. Breakpoints are usually unique to the program that's being debugged. When the user switches programs, they most likely want to use a different set of breakpoints than their old ones. This option ensures that the old breakpoints are disabled and are not used for the new program. This option is enabled by default.

- **Halt at program termination (requires setting a breakpoint):**
  By default, if your program has been linked using a TI runtime library (rts*.lib), a breakpoint is set at the end of the program (C$$EXIT) when the program is loaded. This option allows you to choose not to set the End of Program breakpoint. This option is enabled by default.

  The End of Program breakpoint is used to halt the processor when your program exits following completion. The End of Program breakpoint is not needed if your program does not terminate, however Code Composer Studio will set one anyway as long as this option is set.

  When End of Program code is loaded in RAM, Code Composer Studio sets a software breakpoint. However, when End of Program code is loaded in ROM, Code Composer Studio uses a hardware breakpoint. Since most processors support only a small number of hardware breakpoints, using even one can have a significant impact when debugging.

  Note: You can also avoid using the hardware breakpoint when End of Program code is loaded in ROM by embedding a breakpoint in your code and renaming the label C$$EXIT to C$$EXITE$$ to indicate that this is an embedded breakpoint.

- **Enable CIO function use (requires setting a breakpoint):**
  The C I/O breakpoint is necessary for the normal operation of C I/O library functions such as printf and scanf. The C I/O breakpoint is not needed if you do not want to generate any output, and do not need to read any input. For example, this breakpoint is used in the case of 'printf' to capture the text being printed and display it to the user in CCS. If not set, printf will not do anything. This option is enabled by default. If no code uses C I/O routines, the optimizer should remove library functions and the option will not have any effect.

  When C I/O code is loaded in RAM, Code Composer Studio sets a software breakpoint. However, when C I/O code is loaded in ROM, Code Composer Studio uses a hardware breakpoint. Since most processors support only a small number of hardware breakpoints, using even one can have a significant impact when debugging.

  Note: You can also avoid using the hardware breakpoint when C I/O code is loaded in ROM by embedding a breakpoint in your code and renaming the label C$$IO$$ to C$$IOE$$ to indicate that this is an embedded breakpoint.

Program Verification Options

When a program is loaded, Code Composer Studio can verify (by reading back selected memory) that the program was loaded correctly.

- **Full verification: Verifies everything that has been written to memory**
- **Fast verification: Verifies one word per section (Default Selection)**
- **No verification: No verification will be done when program is loaded**

Realtime Options

- **Halt the target before any debugger access:**
  When the target is running and debugger requires access to the target (e.g. user clicks on refresh button of a view while the target is running), the target will halt for a very brief moment to retrieve the requested data before it continues running again. During this time, time-critical interrupts will be ignored. This option is disabled by default.

- **Enable silicon real-time mode (service critical interrupts when halted, allow debugger accesses while running):**
  This option is supported on only a selected number of emulators. When enabled, debugger accesses will be allowed while the target is running. Time-critical interrupts

are still serviced when the target is halted. This option is disabled by default.

- **Enable polite real-time mode:**
  This option will only be visible when silicon real-time mode is selected. When enabled, CCS will prevent the target from being halted while the application is servicing a time-critical interrupt. If a debugger access requires the target to be halted when servicing a time-critical interrupt, the user will be asked whether Rude Real-Time mode should be enabled. In Rude Real-Time mode, the user regains control of the target while time-critical interrupts are ignored.

Auto Run Options

This option can be used to specify when to halt the program on a program load, restart, or target reset. By default, the program halts at 'main' on a program load or restart.

- **Run to:**
- **On a program load or restart**
- **On a reset**

Launch Options

- **Connect to the target on debugger startup:**
  This option only applies to emulators. When enabled, it automatically connects to the target when a project-less debug session is started. This option is disabled by default.

- **Restore breakpoints from previous session:**
  By default, this option is enabled when debugging the active project. However, it's disabled for project-less debug sessions. When a project-less debug session is terminated and re-launched, all the breakpoints that were previously created will be disabled until the user clicks Target->Restore Debug State. Enabling this option will automatically enable the breakpoints as soon as the project-less debug session is launched. It's recommended to keep this option disabled for project-less debug sessions to avoid the overhead of resolving the breakpoints when a debug session is started.

Connection Options

- **Reset the target on a connect:**
  Automatically resets the target when it's connected. This option is disabled by default.

- **Remove remaining debug state at connect:**
  In some cases, CCS is not able to remove all breakpoints when disconnecting. Enabling this option causes CCS to reattempt to remove any software breakpoint opcodes from memory at connect. This reattempt causes the IDE to access memory (and could potentially put targets in a bad state if that memory is no longer valid). If a target connect is failing due to a silicon bug, disabling this option may let you connect to the target more reliably. This option is enabled by default.

- **Auto connect if a child connects:**
  This option is only visible for routers. When this option is enabled, and the developer connects to a child item in the debug hierarchy, CCS automatically connects to all the ancestors of the child.

OS Aware Debug Options

For more information regarding OS aware debug options, please see the wiki page at: http://wiki.davincidsp.com/index.php?title=Linux_Aware_Debug

- **Display processes and threads in the debug view:**
  When enabled, all the processes and threads will be displayed in debug view. Disabling this option will make the debugger run a little bit faster. However, this option should be enabled if the developer wishes to set software breakpoints, load module symbols, and/or see a list of processes. This option is enabled by default.

- **Update process list on every halt:**
  Updates the list of processes every time the target is halted. Disabling this option may make the debugger update faster on slower emulators.

- **Automatically load module symbols:**
  - **Never (Default selection)**

  - **When modules are loaded and unloaded:**
    This option requires the kernel to be patched. Debugger automatically sets hidden breakpoints.

  - **Whenever halted in the module load/unload routines:**
    Should be used when the kernel cannot be patched. Breakpoints have to be set manually. The target will not automatically be re-run in this case, unless the breakpoints are manually configured to not halt

- **Simulators will flush the pipeline on a halt:**
  This option applies to cycle-accurate simulator. This option should be enabled to make debugging easier and it should be disabled to make profiling more accurate. This option is enabled by default.

- **Automatically step over functions without debug information when source stepping:**
  By default, when stepping into functions that have no source code available, the debugger will automatically step over them. This option is enabled by default.

- **Reset the target on a program load or restart:**
  Resets the target when a program is loaded or restarted. This option is disabled by default.

- **Remember My Settings button:**
  This button is used to save the options back into the launch configuration. If the developer wishes to launch their debug session with the options selected in the Control Panel view, they can save their settings by clicking this button.

See Also:

- Gel Files Overview
- Memory Map Overview
- ARM Advanced Features Overview
- On-Chip Flash Overview

Submit Code Composer Documentation Feedback

2.3.2. Gel Files Overview

GEL files view allows you to examine and manage GEL files that are loaded in debugger for CPU that is selected in Debug View. Loaded GEL files will be displayed only after debugger is started.

User may double click on any gel file to open it in editor and make further edits. User will need to reload the GEL file after modification in order for changes to take effect in debugger.

Commands available from context menu:

Open - Open GEL file in text editor

Reload - Allows user to reload a GEL file after modifications are performed.

Load - Allows user to load a GEL file explicitly during a debug session. The file will need to be loaded every time a debug session is started/terminated. The best way is to add it to target configuration file being used (using setup editor) or if one already is specified, then add it to that files StartUp function with GEL_LoadGel function.

Remove - Remove currently selected GEL file from debugger.

See Also:

- Debugger Options Overview
- Memory Map Overview

- ARM Advanced Features Overview
- On-Chip Flash Overview

Submit Code Composer Documentation Feedback

2.3.3. Memory Map Overview

The memory map tells the debugger which areas of memory it can and cannot access. In addition on some devices it may specify additional parameters that tell debugger how to access memory. Memory maps vary depending on the device/application. Typically, the memory map matches the MEMORY definition in your linker command file, which should match device that application will run on.

Memory Map view provides a read only view of memory map as it is defined in debugger. Memory map information is displayed only after debugger has been started.

When you first invoke Code Composer Studio, the memory map is turned off. You can access any memory location; the memory map does not interfere. When memory mapping is enabled, the debugger checks each of its memory accesses against the memory map provided. If you try to access an undefined or protected area, the debugger will prevent the access from occurring and the debugger will display the value as a series of dashes (-).
Note: Device simulators use hard-coded memory map settings that match closely memory as it is defined in real HW.
Accessing Nonexistent Memory When the debugger compares memory accesses against the memory map, it performs this checking in software, not hardware. The debugger cannot prevent your program from attempting to access nonexistent memory.

Memory map may be defined using a GEL file, which is then specified in target configuration. When debugger is launched with that target configuration, then GEL file is loaded and memory map will be automatically defined. GEL provides a complete set of memory-mapping functions. The easiest method of implementing a memory map is to put the memory-mapping functions in a GEL text file and execute the GEL file at startup.

Memory Map view provides a convienent link to currently loaded GEL files. You may also select Tools->Gel Files to see currently loaded gel files. Double clicking on GEL file will open it in editor.

See Also:

- Debugger Options Overview
- Gel Files Overview
- ARM Advanced Features Overview
- On-Chip Flash Overview

Submit Code Composer Documentation Feedback

2.3.4. Code Composer Studio Help > Views and Editors > Control Panel View

On-Chip Flash Overview

The On-Chip view allows you to control Flash memory related settings, as well as the ability to perform Flash memory related operations. Settings for these options are used for controlling how Flash memory related operations are performed; this includes the operations available in this view (via buttons), as well as controlling the way the Debugger handles writing to Flash memory when performing a program load. The options and operations available in this view are dependant on the connected target.

Supported Targets:

- TMS320F28x
- Stellaris Cortex-M3
- TMS570

Supported Settings/Operations for TMS320F28x:

- **Clock Configuration:**
  The clock configuration setting is used to calculate the System Frequency for Flash operations.

- **Flash Program Setting:**
  Determine how the debugger handles Program Load operations (Erase/Program/Verify/RAM Only).

- **Erase:**
  Erases Flash Memory, which can be performed on the entire Flash (default), or only on specific Flash sectors.

- **Code Security Password - Program Password:**
  The eight 16-bit passwords (Key 0 - Key7) form a 128-bit password that the Code Security Module (CSM) uses to lock the device. Key 7 is located at address 0xAE7 in program space and is the most-significant word. Key 0 is located at address 0xAE0 in program space and is the least-significant word.

  New devices are unsecured. An unsecured device has an unsecured password. An unsecured password has values at the password locations PWL0 - PWL7, (0x3F7FF8 - 0x3F7FFF), which are all 0xFFFF. You can read and program flash memory on an unsecured device. You cannot lock an unsecured device. In order to secure, or lock, a device, it must have a secure password.

  A secure device has a secure password. A secure password has one or more of the password values at the password locations PWL0 - PWL7, (0x3F7FF8 - 0x3F7FFF), not equal to 0xFFFF.

  Once you program a secure password, the device locks on the next power-cycle (power-down, power-up). A secure device that is locked does not allow you to read or write flash memory. You must unlock the device to read or write flash memory.

- **Code Security Password - Lock:**
  To lock a secure device, click the Lock button.

  Locking a device does not require the password. The On-Chip Flash Programmer locks a secure device by writing to the Code Security Module Status and Control Register (CSMSCR) at address 0xAEF to set the FORCESEC bit (bit 15). When a secure device is locked, you will see zero data values appear in all flash memory locations.

- **Code Security Password - Unlock:**
  To unlock a secure device, enter the 128-bit password into the Key 0 - Key 7 fields, then click the unlock button.

- **Frequency Test**
  Pressing on the Start Frequency Test button will start the freuqueny test operation. This operation toggles the specified GPIO Register pin to confirm proper clock configuration; and requires you to monitor the specified GPIO Register pin with an oscilloscope.

  Frequency Testing leaves the target in a running state; which allows you to measure the frequency during this time. For correct clock configuration, the measured frequency should read 10kHz (100uS +/- 10uS cycle time). Otherwise, the clock configuration settings might be incorrect and should be corrected before starting any Flash operations.

  To stop the Frequency test, press the End Frequency Test button.

- **Depletion Recovery:**
  The depletion recovery algorithm looks for sectors that are in depletion and attempts to recover them. All sectors on the device are checked.

- **Checksum:**
  The Calculate Checksums flash operation calculates checksums for these types of memory: Flash memory & One Time Programmable (OTP) memory.

  The Flash memory and OTP memory checksums are calculated on the target as a 16-bit sum by adding together all 16 bit memory locations in the flash or OTP range (the lower 16bit result of the summation is returned).

- **Program Load:**
  When a user loads a program containing sectors in Flash memory, the debugger will first erase the Flash (based on the erase settings), and then load the flash sectors to Flash memory.

Supported Settings/Operations for Stellaris Cortex-M3:

- **Crystal Frequency:**
  Modify the crystal frequency for Flash operation timing.

- **Erase:**
  Erases Flash Memory, which can be performed on:
  1. **The Entire Flash (default)**
  2. **Necessary Pages Only; only available during a Program Load operation. This will erase only the pages that the current program writes to; the remaining pages will retain their data**
  3. **By Address Range. Given a start and end address in Flash memory, this option will allow users to erase only the Flash pages associated with the address range. Note: The specified address range will be mapped to erasable pages on the target, and depending on the size of a Flash page, additional memory addresses could be erased compared to the user specified address range.**

- **Program Load:**
  When a user loads a program containing sectors in Flash memory, the debugger will first erase the Flash (based on the erase settings), and then load the flash sectors to Flash memory.

Supported Settings/Operations for TMS570:

- **Enable Programming to OTP Memory:**
  Enabling this option will allow Program Load to write to OTP Flash Memory. As OTP memory cannot be erased, this option is off by default.

- **Crystal Frequency (Mhz):**
  Use for calculating System Frequency and PLL values for Flash operation timing.

- **Erase:**
  Erases Flash Memory, which can be performed on:
  1. **The Entire Flash (default)**
  2. **Necessary Sectors Only; only available during a Program Load operation. This will erase only the sectors that the current program writes to; the remaining sectors will retain their data**
  3. **Selected Sectors Only; this will allow the user to choose the Banks and Sectors (available on the target) they wish to erase.**

- **Program Load:**
  When a user loads a program containing sectors in Flash memory, the debugger will first erase the Flash (based on the erase settings), and then load the flash sectors to Flash memory.

See Also:

- Debugger Options Overview
- Memory Map Overview
- ARM Advanced Features Overview
- Gel Files Overview

Submit Code Composer Documentation Feedback

2.3.5. ARM Advanced Features Overview

Description of ARM Advanced Features

Configuring TMS470Rx Hardware

This document shows you how to use the advanced features of the TMS470Rx processor. This document assumes that you are familiar with Code Composer™. The following features are specific to the TMS470R2x and TMS470R3x processors. These features are activated on the ARM analysis interface only if your PC is connected to those processors. You can enable/disable these features according to your specifications.

ARM9/11 Specifics:

MMU Enabled -
Alignment Fault Checking enabled
Data Cache enabled
Instruction Cache enabled
Write Buffer enabled B
reak on Reset
Break on Undefined Instruction
Break on SWI Break on Program Abort
Break on Data Abort
Break on IRQ
Break on FIQ

Cache Management:

Flush Instruction Cache
Flush Data Cache
Flush All Caches
Clean Data Cache

TLB Management:

Flush Instruction TLB
Flush Data TLB
Flush All TLBs

See Also:

- Debugger Options Overview
- Memory Map Overview
- Gel Files Overview
- On-Chip Flash Overview

Submit Code Composer Documentation Feedback

2.4. Pin Connection Tool - Simulating External Interrupts

The simulator allows you to simulate and monitor external interrupt signals.

The Pin Connect tool enables you to specify the interval at which selected external interrupts will occur.

To simulate external interrupts:

1. **Start a project-less debug session by clicking on the 'Launch TI Debugger' button located in the workbench toolbar**

2. **Create a data file that specifies interrupt intervals.**
3. **Start the Pin Connect tool. From the Window -> Show View menu, choose Pin Connect.**
4. **Connect your data file to an external interrupt pin.**
5. **Load your program.**
6. **Run your program.**

See Also:
Creating a Data File
Connecting to a Data File
Repetition of Patterns for a Specified Number of Times
Repetition to the End of Simulation
Relative Clock Cycle
Absolute Clock Cycle

2.4.2. Setting up Your Input File

To simulate external interrupts, you must first create a data file that specifies interrupt intervals. Interrupt intervals are expressed as a function of CPU clock cycles. Simulation begins at the first clock cycle. An interrupt will occur at each specified clock cycle.

Your data file must contain a CPU clock cycle parameter in the following format:

clock cycle [**rpt** { n| **EOS**}]

clock-cycle	The CPU clock cycle parameter specifies the intervals at which interrupts will occur. Clock cycles can be specified as absolute or relative.
rpt	Repeat the same pattern a fixed number of times.
n	A positive integer value, specifying the number of times to repeat.
EOS	Repeat the same pattern until the end of simulation.

See Also:
Overview of the Pin Connection Tool
Connecting to a Data File
Repetition of Patterns for a Specified Number of Times
Repetition to the End of Simulation
Relative Clock Cycle
Absolute Clock Cycle

2.4.3. Connecting a Data File

After creating your data file, you must connect the file to an external interrupt pin.

To Connect a Data File to an External Interrupt Pin

1. **Select Window -> Show View -> Pin Connect from the Code Composer Studio menu bar. The Pin Connect tool appears. The available external interrupt pins are listed in the Pin Name column.**
2. **Click on the Pin Name that you wish to use. Notice that the Filename column displays the status "Not Connected".**
3. **Click the Connect button (or double-click on the selected Pin Name). The Open Pin File dialog box appears. Note that if the connect button is disabled, it either means that a target is not selected in debg view or the selected target doesn't support pin connection.**
4. **Specify the name of your data file in the File name field.**
5. **Click the Open button.**

The selected pin is connected to your data file. Notice that the selected data file is now listed in the Filename column.

See Also:
Overview of the Pin Connection Tool
Creating a Data File
Repetition of Patterns for a Specified Number of Times
Repetition to the End of Simulation
Relative Clock Cycle
Absolute Clock Cycle

2.5. Overview of Port Connect Tool

You can use the Port Connect tool to access a file through a memory address. Then, by connecting to the memory (port) address, you can read data in from a file, and/or write data out to a file. This tool is essentially used to simulate reading from and writing to a port.

See Also:
Connecting a File
Disconnecting a File

2.5.2. Connecting a File

To connect a memory (port) address to a data file, follow these steps:

1. **Start a project-less debug session by clicking on the 'Launch TI Debugger' button located in the workbench toolbar**
2. **From the Window menu, select Show View, Port Connect. This action displays the Port Connect view.**
3. **Click the Connect button. This action opens the Connect dialog box. Note that if the connect button is disabled, it either means that a target is not selected in debg view or the selected target doesn't support port connection.**
4. **In the Port Address field, enter the memory address. This parameter can be an absolute address, any C expression, the name of a C function, or an assembly language label. If you want to specify a hex address, be sure to prefix the address number with 0x; otherwise, Code Composer Studio treats the number as a decimal address.**
5. **In the Length field, enter the length of the memory range. The length can be any C expression.**
6. **In the Page combo box (if available), choose type of memory that the address occupies.**
7. **In the Type field, select the Write or Read radio button. By default, the Read operation will loop through the data file to continuously read in information from the data file. You have the option of disabling this by selecting the "No Rewind" checkbox. This will force it to stop when it hits the end of the data file.**
8. **Click OK. This action displays the Open Port File window.**
9. **Select the data file to which you want to connect and click Open.**
10. **Load your program.**
11. **Run your program.**

The file is accessed during an assembly language read or write of the associated memory address. Any memory address can be connected to a file. A maximum of one input and one output file can be connected to a single memory address; multiple addresses can be connected to a single file.

See Also:
Overview of the Port Connection Tool
Disconnecting a File

2.5.3. Disconnecting a File

Before you can delete from the memory map a memory address that you have connected to a file, you must disconnect the address. To disconnect a memory address from a data file, follow these steps:

1. **Select the line that contains the file and port address you want to disconnect. This action activates the Disconnect button.**
2. **Press the Disconnect button. This action disconnects the memory address from the data file and removes that information from the Port Connect view.**

See Also:
Overview of the Port Connection Tool
Connecting a File

2.6. Profile Setup View

The Profile Setup View configures a profiling task, and hence controls what profiling data is collected and how a profiling is run. For standard profiling, the user selects activities from the list of profiling activities. Further, the user can customize activities through properties, create various profiling configurations for storing customized activities, and control when a profiling task pauses collecting data and resumes collecting data through setting up the control points.

ToolBar

- **New -** Create a new configuration
- **Delete -** Delete the selected configuration.
- **Copy -** Create a configuration from the copy of the selected configuration
- **Minimize -** Minimize the view
- **Maximize -** Maximize the view

2.7. Scripting Console View

The Scripting Console View allows you to enter JavaScript commands, such as Debug Server Script and GUI Script. The view emulates a shell environment, providing command history, auto-complete, command editing and command navigation. <TAB> key may be pressed to get a list of supported commands. Typing “help command” or “command /?” will display command specific help including parameter information.

The following is a list of available actions in this view:

Image	Name	Description	Availability
	Clear Console View	Clear the content of the scripting console view, this action is the same as the *cls* command.	View action
	Copy (CTRL+C)	Copy the selected text from the scripting console view to the clipboard.	Context Menu
📂	Load Command File	Load a command file from disk and execute it immediately, this action is the same as the *execCmdFile* command.	View action
	Reset to Default	Reset the command group filter to the default state.	View action
	Paste (CTRL+V)	Paste the text from the clipboard to the scripting console view.	Context Menu
	Pin Active Debug Session	Pin the debug server scripting commands to the active debug session that you have selected in the Debug View; it is useful for multiple CPU debug dession.	View action
	Scripting Console Commands GUI Commands DSS Commands others...	Select the command group(s) that you want to see when pressing <TAB> key in the scripting console view.	View action

Commands

The scripting supports various commands group, internal command group (Scripting Console Commands), GUI Commands, Debug Server Commands, other (Unspecified Commands). You can select the command filter in the view local toolbar, this will allow a short list of command to be display when the <TAB> key is pressed. You can contribute commands to the scripting console view by creating a set of functions in a JavsScript file, than use the loadJSFile (see below) command to add the functions to as a command.

Internal command group provides a set of command internally available in the scripting console view. The following is a list of internal commands:

- **cls clear the scripting console content.**
- **help display help content for the specified command, if there is any.**
- **loadJSFile load a JavsScript file or all files within a directory.**
- **print echo the result of the provided expression.**
- **services display a list of top level service provider. A service provider is an entry point to a native Java scriptable object, i.e gss_0, ds, env, etc...**
- **setWindowBuffer set the number of buffer line for the scripting console view.**
- **unloadJSFile unload a JavaScript file or all files within a directory. You will need to restart CCS to completely unload the script from memory.**
- **enableLog enable logging of input, output, and error to a file.**
- **disableLog disable logging to a file.**
- **execCmdFile execute a command file. You can create a command file as what you would type in the scripting console manually and store it in a text file, than use the execCmdFile command to load the file and immediately execute the command(s). enableLog command comes in handy to log the commands that you entered in the console to a file and than execute it using the execCmdFile command.**

GUI command group provides a set of UI specific commands that interacts with user interface, such as *cleanProject*, *buildProject*, etc... These commands will execute in the current workbench window context, if you have more than one workbench window open, it won't affect the other workbench window.

Debug Server command group provides a set of debug server commands that uses the debug server scripting (DSS) api. These commands will interact with the current debug context in the workbench window, unless you have the active session pinned using the toolbar action.

Unspecified command group contains a list of commands that doesn't have any associtated group (or service). The group is provided as part of the function documentation of a JavaScript file.

JavaScript Command File

You can contribute commands to the scripting console view by creating a JavaScript file and load the file with the *loadJSFile* command. You can also provide documentation for your functions with a JavaDoc like syantax, the information will be shown when you execute the help command for the specified function.

Example:

```
/**
 * Read multiple words (of the default word-size) from the target and return the result as an array of integers.
 * @param address the starting.
 * @param page a one-digit number that identifies the type of memory: 0 - Program, 1 - Data, 2 - I/O
 * @param numWords number of words to read.
 * @param signed whether the returned words are signed or unsinged.
 * @return an array of target words.
 * @service DSS Debug Server Commands
 */
function readWord(address, page, numWords, signed) {
  return activeDS.memory.readWord(page, address, numWords, signed);
}
```

Submit Code Composer Documentation Feedback

2.8. Memory View

The memory view helps you to view the contents of memory starting at a specified address. You can enter a starting address in the Address Text box at the top of the view. You can also edit the contents of a selected memory location by double-clicking the value. You can display multiple instances of the memory views. Any property changes made within an open memory view apply only to that memory view.

These options are available.

Command	Name	Description	Availability
	Save	Save the memory data to a file.	View action/Context menu
	Load	Load the memory data from a file.	View action/Context menu
	Fill	Fill the memory with a data value.	View action/Context menu
	Enable/Disable Memory Analysis	Enable or Disable the Memory Analysis feature if the target supports it.	View action
	Enable/Disable Cache Coloring	Show or Hide the cache coloring if the target supports cache memory.	View action/Context menu
	Enable/Disable Cache Line Markers	Show or hide the Cache Line markers if the target supports cache memory.	View action/Context menu
	Show Physical View/CPU View	Available on some ARM targets (ARM9 only) that allow the memory view to be toggled between displaying CPU and physical memory. Virtual memory can be accessed when the ARM processor's MMU (Memory Management Unit) is enabled.	View action
	Refresh	Refreshes the contents of the view.	View action/Context menu
	Continuous Refresh	Continuously refreshes the contents of the view.	View action
	Pin To Context	Forces the memory view to keep on displaying data for the selected debug context even though the selected context is no longer the current active context.	View action
	Lock Memory View	Stop the memory view from automatically refreshing when memory changes.	View action/Context menu
	New Memory View	Opens a new memory view.	View action
	Customize Colors And Fonts	Sets the colors and fonts in Memory View..	View action
	Configure	Set preferences for the memory view.	Context menu
	Copy to Clipboard	Copies the selected cells or all visible cells to the system clipboard, optionally include addresses.	Context menu
	Copy to Memory	Copies the selected cells or all visible cells to another memory location.	Context menu
	Print	Prints the selected cells or all visible cells.	Context menu
	Reference Buffer	Sets up, enables and captures the reference buffer.	Context menu

Submit Code Composer Documentation Feedback

2.8.2. Working with Memory

Inspecting and editing the memory

You can inspect and change the memory.

- To view and/or edit the memory, enter a starting address in the Address Text box at the top of the view. If the target supports various memory pages, the Memory Page dropdown menu is enabled and you can select a memory page from where memory data should be read.

- To edit the content of a selected memory location, double-click your cursor inside the cell corresponding to the address you wish to edit and the value will become editable. When you move the cursor elsewhere, the value will be saved and highlighted in red to show it has been changed.

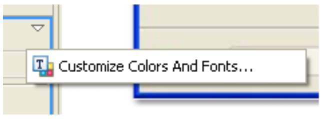
The **Memory** view supports the same addressing as the C and C++ languages. You can address memory using expressions such as:

- `0x0847d3c`
- `(&y)+1024`
- `*ptr`

Changing fonts and colors

You can customize the fonts and colors displayed in the Memory view.

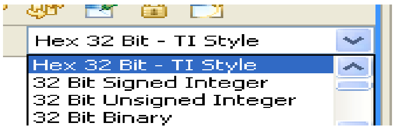
1. Select a display format from the **Display Format** dropdown menu in the toolbar of the Memory View.



Changing displayed format

You can configure your output to display hexadecimal, ascii, binary, character, and/or signed/unsigned decimal. You can configure each memory view independently.

1. Select a display format from the **Display Format** dropdown menu in the toolbar of the Memory View.


Hex 32 Bit - TI Style

Hex 32 Bit - TI Style
32 Bit Signed Integer
32 Bit Unsigned Integer
32 Bit Binary

Submit Code Composer Documentation Feedback

2.8.3. Default Fonts And Colors In Memory View

1 2 3 4	Normal valid data
1 2 3 4 (red)	Memory value has changed since the last time the memory window was updated.
1 2 3 4 (bold)	Data has been flagged by one or more of the memory analyzers that have been configured in the memory windows properties. To determine why the address is flagged, click the item in the memory window and open the properties for the memory window. Details about the attributes of the address that was clicked on are displayed in the properties. **Note:** You must specifically enable the Memory Analysis feature in the Property Window before this feature can be used.
1 2 3 4 (gray italics)	Processor is running, data cannot be modified in the memory window.
1 2 3 4 (grayed out data)	Read-only memory.
XXXX (uppercase Xs)	Memory does not exist For example, a C64XX+ device has the Background Coloring Mode property for a memory window configured as Memory Level Coloring and all memory levels are bypassed by clearing the check boxes at the bottom of the memory window. XXXX displays at addresses where L2 memory exists in the device memory map and no external memory is mapped at that address because the L2 memory is being bypassed.
---- (dashes with gray background)	No data available (for example, target is disconnected)
… (ellipsis)	Display is being scrolled—data not requested
??? (question marks):	Error reading memory location
*** (asterisks)	Non-emulatable memory location (that is, secure code). **Note:** The disassembly window also displays non-emulatable memory locations as ****.

Note: The number of characters displayed depends on the data type of the configurated memory window.

Submit Code Composer Documentation Feedback

2.8.4. Memory view preferences

You can change the following display characteristics of the memory view. Changes made to these characteristics are applied to the system level.

- Cache Line Boundary Marker Color—color of the cache line boundary markers
- Memory Cell Background Color—background color of the table cells in the Memory View
- Memory Cell Edit Color—color of the text that has been modified but not yet been saved
- Memory Cell Selection Color—background color of the memory cells that have been selected
- Memory Cell Text Color—color of the text displayed in the memory cells
- Memory Cell Text Font—font of the text displayed in the memory cells
- Memory Level Colors—background color to indicate the cache level for the data being displayed

Submit Code Composer Documentation Feedback

2.8.5. Saving and Loading Memory Data Files

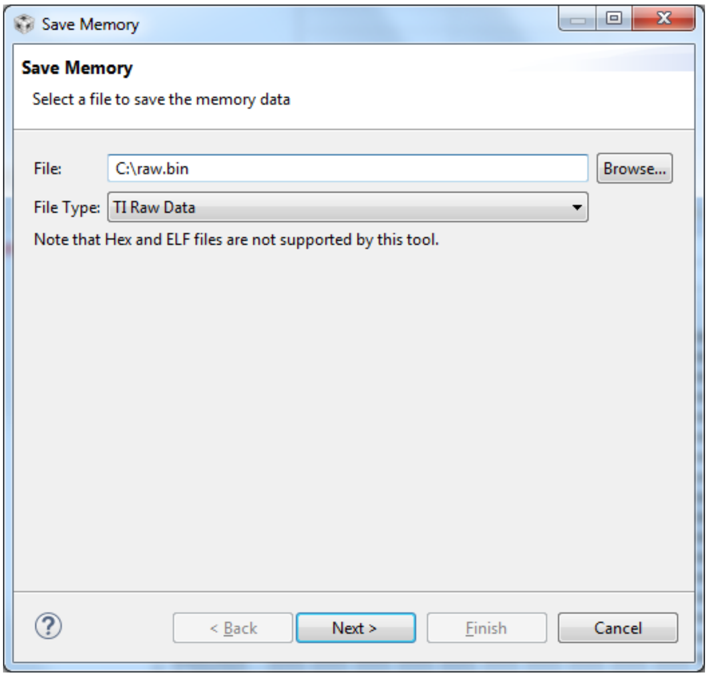
In general, these wizards help users to Save Memory data to a file in a supported format and to Load Memory data into the memory from a file that was produced from the Save Memory feature. This tool allows you to support raw binary data transfers, and also allows for faster download speeds. This new option is available on all targets. There is also an option to perform byte swapping for every 16-bit word during the data transfer.

Save Memory Wizard

Use the Save Memory wizard to save the memory data to a specified file. You can also specify the start address and length of the memory block.

1. Open Memory view.

2. Right-click in the Memory view and select **Save Memory** command from the context menu. The wizard displays.



Save Memory

Select a file to save the memory data

File: C:\raw.bin

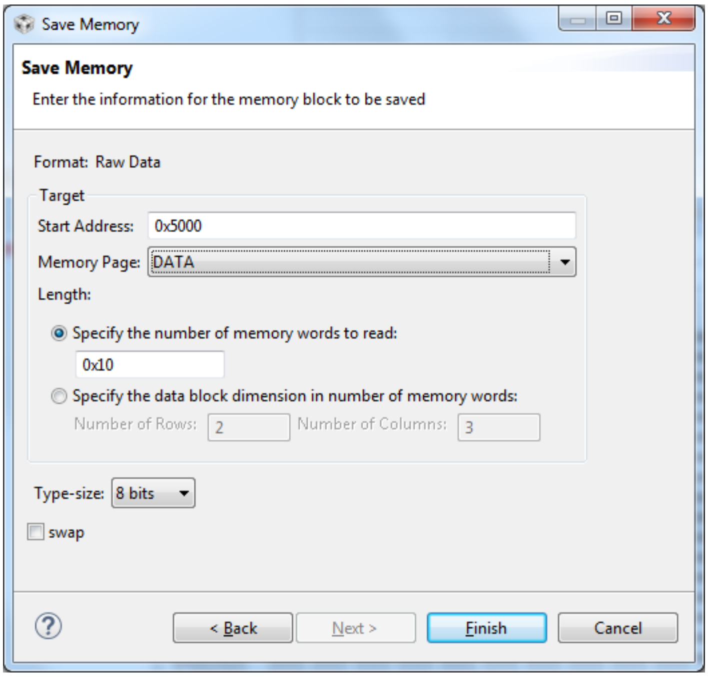
File Type: TI Raw Data

Note that Hex and ELF files are not supported by this tool.

3. Browse for the desired save file or create a new file name using the **Browse** button. A File Open dialog will be displayed so users can select a file by navigating through the file system. You can select the Files of Type list to save the memory as TI Data Format (`*.dat`), Coff Format (`*.out`), or Raw Data Format (`*.bin`).

4. Click **Next**.

5. Select information for the memory block to be saved, including the format, start address, and length. Some targets may have a Page field that indicates the page of the memory block to be saved. If the Raw Data file type was selected, the Type size field specifies the data boundary when interpreting data as Little Endian or Big Endian. These options are saved in the file header.



Save Memory

Enter the information for the memory block to be saved

Format: Raw Data

Target

Start Address: 0x5000

Memory Page: DATA

Length:

● Specify the number of memory words to read:

0x10

○ Specify the data block dimension in number of memory words:

Number of Rows: 2 Number of Columns: 3

Type-size: 8 bits

□ swap

Save Memory Option	Description
Format	- `*.dat` TI formats are Hex, Integer, Long, Float and Addressable Unit. The selection defaults to Hex. - `*.out` filename extensions require the Format field be set to `Coff` and not changed by users. - All other file name extensions require the Format field be set to Raw Data and cannot be changed.
Start Address	The memory block starting address that is entered as a valid memory address or an expression.
Memory Page	Displays only when the target supports more than one memory pages and indicates the page of the memory block to be saved. Select a page from the list. The selection defaults to the first page.
Length	Memory block size in number of memory words to be saved. It can be entered as a decimal or hexadecimal value.
Type-size	- Enabled for Raw Data Format. - Specifies the data boundary when interpreting data as Little Endian or Big Endian. - Select 8 bits, 16 bits, 32 bits, or 64 bits.
Swap	Check box is enabled only for the Raw Data Format and can perform byte-swapping for every 16-bit

	word.

6. Click **Finish**. Your data is saved to the specified file.

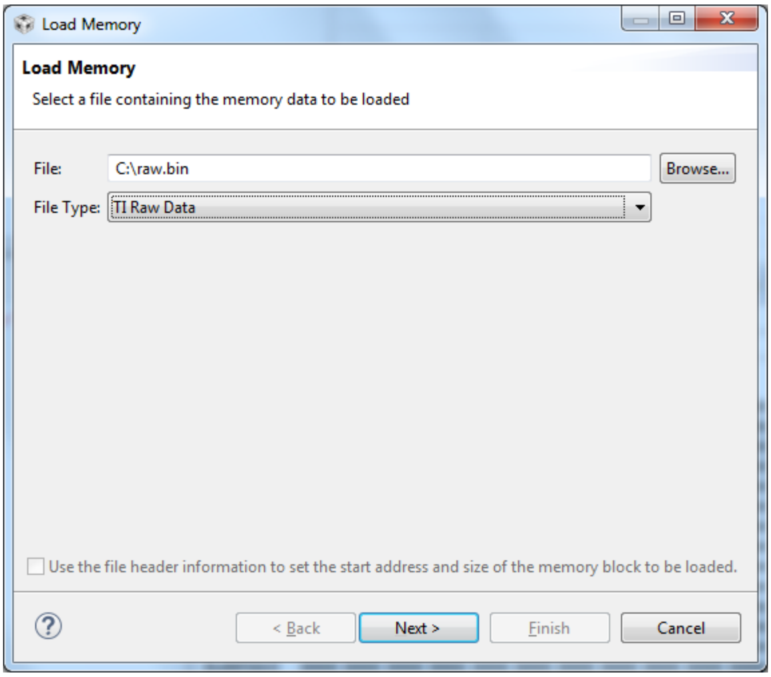
Note: All user options are retained and available when the wizard is reused.

Load Memory Wizard

You can use the Load Memory wizard to load a memory file from the directory you specify. You can also choose the start address of a memory block to load that memory file.

1. Open Memory view.

2. Right-click on the Memory View and select **Load Memory** from the context menu.



Load Memory

Load Memory

Select a file containing the memory data to be loaded

File: C:\raw.bin

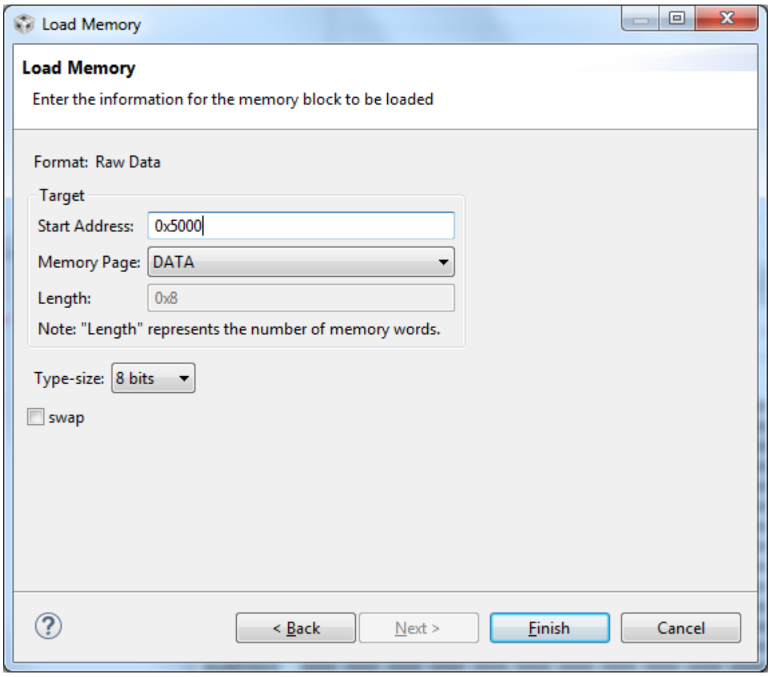
File Type: TI Raw Data

□ Use the file header information to set the start address and size of the memory block to be loaded.

3. Browse for the desired memory data file and click **Open**.

4. Select the **Use the file header...** check box to have the file header information used to set the starting address and size of the memory block you want to load by selecting that option on this page if you are loading a `*.dat` or `*.out` file. This populates the fields on the next graphic that displays when you click **Next.**

5. Alternatively, you can click **Next** to review or modify the memory block information before loading the file.



Load Memory

Load Memory

Enter the information for the memory block to be loaded

Format: Raw Data

Target

Start Address: 0x5000

Memory Page: DATA

Length: 0x8

Note: "Length" represents the number of memory words.

Type-size: 8 bits

□ swap

6. Enter the information for the memory block to be loaded, including the format, start address, length, and type size, if applicable.

Load Memory Option	Description
Start Address	The memory block starting address to be loaded that is entered as a valid memory address or an expression.
Memory Page	Displays only when the target supports more than one memory pages and indicates the page of the memory block to be loaded. Select a page from the list. The selection defaults to the first page.
Length	Memory block size in number of memory words to be loaded. Entered as a decimal or hexadecimal value.

	If the filename extension is not `*.dat` or `*.out`, Code Composer treats the file as Raw Data format and this field displays the file size. In this situation, the field cannot be changed.
Type-size	• Enabled for Raw Data Format. • Specifies the data boundary when interpreting data as Little Endian or Big Endian. • Select 8 bits, 16 bits, 32 bits, or 64 bits.
Swap	Check box is enabled only for the Raw Data Format and can perform byte-swapping for every 16-bit word.

7. Click **Finish**. The file is loaded to the target memory.

Note: All user options are retained and available when the wizard is reused.

2.8.6. Data File Formats

The debugger reads and writes data using these formats:

- Common Object File Format (COFF)—binary
- Code Composer data file format—text

See the Saving and Loading Memory Data Files topic for more information on using data files for memory loads.

Common Object File Format (COFF)

A binary file that uses the Common Object File Format (COFF). This is the most compact way of storing large blocks of data on the PC.

Code Composer data file

A text file that contains one line of header information and stores the data as one sample per line. The legacy file format supported these formats :

- Hexadecimal
- Integer
- Long
- Float

The new scheme which is backwards compatible supports a larger set of data types. GEL_MemoryListSupportedTypes() GEL API could be used to list complete set of supported data types in the new scheme

The header information for data files uses this syntax:

```
MagicNumber  Format StartingAddress PageNum Length [NewFormat]
```

- `MagicNumber`—fixed at `1651`.
- `Format`—a number from 1 to 4, indicating the format of the samples in the file.
  This number represents a data format: (`1`) hexadecimal, (`2`) integer, (`3`) long, (`4`) float, or (`9`) Use new scheme
- `StartingAddress`—starting address of the block that was saved.
- `PageNum`—page number the block was taken from.
- `Length`—number of samples in the block.
- `NewFormat`—Format (9); the new scheme. This is optional when usign the legacy formats 1 - 4

> **Format 9) indicates usage of the new scheme. This was added to achieve consistency in the formats available in the memory browser and the formats available with load/save feature. The memory browser supports a larger set of data formats which vary slightly across each target. The old load/save header is based on a fixed set of data formats. In order to maintain backwards compatibility the existing fixed data format is still supported. However, an additional data format (9) was introduced which acts as a switch to use the new scheme. The last extra parameter (NewFormat) is the actual format chosen in the new scheme .**

All header values are assumed to be TI-style hexadecimal values.

This is an example of a Code Composer data file:

```
1651 1 800 1 10
0x0000
0x0000
0x0000
0x0000
0x0000
0x0000
0x0000
0x0000
0x0000
0x0000
0x0000
0x0000
0x0000
```

Note: Code Composer expects five-digit numbers of information to read in 4-digit values. While the data is known to be hexadecimal, Code Composer expects the first digit to be a zero. Code Composer does this so that hexadecimal numbers beginning with letters (that is, `F800`) are not misread as labels. For example, when reading data in hexadecimal format from a data file, the first digit may be truncated.

Look at this input data:

```
0022
0022
0033
8AC
FC94
13
895
AB9 ...
```

When reading this data, Code Composer actually reads data as:

```
0022
0033
08AC
0C94
```

```
0003
0095
00B9 ...
```

But if you test Code Composer IDE with five-digit data input, (by adding zeroes at the beginning where necessary), the output is consistent with the input:

```
00022
00033
018AC
0FC94
00013
00895
00AB9
```

Submit Code Composer Documentation Feedback

2.9. Variables View

This view displays information about the variables in the currently selected stack frame. In addition to its name, the view shows each variable's value, type, and memory address. The display format of a variable's value can be set to Default, Hexadecimal, Decimal, Binary, and Octal.

Variables that contain more than one element, such as arrays, structures, or pointers, are displayed with either a plus sign (+) or minus sign (-) immediately preceding the expression name.

- The (+) symbol indicates that the expression contains elements that can be further expanded.
- The (-) symbol indicates that the expression is fully expanded and can be collapsed to reduce the information displayed.

Variables whose values have changed since the last time they were seen are highlighted in yellow background. Whenever the situation permits it, the value of a variable may be modified by clicking its value column and entering a new value.

Following are the commands available in the view:

Image	Name	Description	Availability
	Change Value	Allows you to change the value for the underlying selected variable.	In-place editing on the Value cell of the variable row
	Cast To Type	Casts a variable to a different type.	Context menu
x[ ]	Display As Array	Display a variable as an array. This is available when the type or casted type of the variable is a pointer.	Context menu
	Restore Original Type	Restores a variable to its original type.	Context menu
	Copy Variables	Copies the selected variables to the system clipboard.	Context menu
	Collapse All	Collapses all the currently expanded variables.	Toolbar action
	Number Format	Selects the display format for the values of individual variables or all variables in the view. The format at an individual variable takes precedence over the format at the view.	To apply format on individual variables, select the desired variables and bring up context menu item Number Format to select a desired format. To apply format on all variables in the view, select Toolbar View Menu and select Number Format menu item to select a desired format.
	Open New View	Opens a new view.	Toolbar action
	Refresh	Refreshes the view.	Toolbar action
	Select Columns	Selects to show or hide columns of the view.	Toolbar View Menu action and select Layout menu item and click Select Columns
	View Memory	Show the memory with the address of the selected variable in a Memory View.	Context Menu
	View Memory at Value	Show the memory with the value of the selected variable in a Memory View.	Context Menu
x+y =?	Watch	Creates an expression for the selected variable in Expressions View.	Context menu

Related tasks

Tracking a Variable's Value

Related reference

Tracking an Expression's Value
Launch Graph from Variables View

Submit Code Composer Documentation Feedback

2.10. Expressions View

This view allows you to specify C/C++ expressions and monitor their values. The C/C++ expressions can be composed of local variables, global variables, static variables, and other C/C++ expressions. You can inspect data from a stack frame of a suspended thread, and other places. In addition to the name of an expression, the view shows each its value, type, and memory address. The display format of the value of an expression can be set to Default, Hexadecimal, Decimal, Binary, and Octal.

Expressions that contain more than one element, such as arrays, structures, or pointers, are displayed with either a plus sign (+) or minus sign (-) immediately preceding the expression name.

- The (+) symbol indicates that the expression contains elements that can be further expanded.
- The (-) symbol indicates that the expression is fully expanded and can be collapsed to reduce the information displayed.

Expressions whose values have changed since the last time they were seen are highlighted in yellow background. Whenever the situation permits it, the value of an expression may be modified by clicking its value column and entering a new value. The view may be configured to refresh automatically in regular intervals.

Following are the commands available in the view:

Image	Name	Description	Availability

	Add Expression	Allows you to add a new expression.	In-place editing on the <Add new expression> cell of the last row
	Edit Expression	Allows you to edit the currently selected expression.	In-place editing on the Expression cell of the expression row
✖	Remove	Removes the selected expression from the view.	Toolbar action
✖	Remove All	Removes all expressions from the view.	Toolbar action
	Change Value	Allows you to change the value for the underlying variable of the expression.	In-place editing on the Value cell of the expression row
()	Cast to Type	Casts an expression to a different type.	Right-click at the expression to bring up its context menu item Cast To Type. Alternatively, prepend the expression with the desired type, e.g., cast to char type in the expresssion i+1: (char)(i+1)
×[ ]	Display As Array	Display an expression as an array. This is available when the type or casted type of the expression is a pointer.	Context menu
	Restore Original Type	Restores an expression to its original type.	If type casting is done by context menu item Cast To Type, select context menu item Restore Original Type. If type casting is done by prepending the expression with a type, remove the prepended casted type from the expresssion.
📋	Copy Expressions	Copies the selected expressions to the system clipboard.	Context menu
⊟	Collapse All	Collapses all the currently expanded expressions.	Toolbar action
	Number Format	Selects the display format for the values of individual expressions or all expressions in the view. The format at an individual expression takes precedence over the format at the view.	To apply format on individual expressions, select the desired expressions and bring up context menu item Number Format to select a desired format. To apply format on all expressions in the view, select Toolbar View Menu and select Number Format menu item to select a desired format.
	Open New View	Opens a new view.	Toolbar action
🔄	Refresh	Refreshes the view.	Toolbar action
🔄	Continuous Refresh	Enables the view to automatically refresh on a regular basis.	Toolbar action
	Customize Continuous Refresh Interval	Sets the refresh interval for Continuous Refresh.	Toolbar View Menu action
	Select Columns	Selects to show or hide columns of the view.	Toolbar View Menu action and select Layout menu item and click Select Columns
	View Memory	Show the memory with the address of the selected expression in a Memory View.	Context menu
	View Memory at Value	Show the memory with the value of the selected expression in a Memory View.	Context menu

Related tasks

Tracking an Expression's Value

Related reference

Tracking a Variable's Value
Launch Graph from Expressions View

Submit Code Composer Documentation Feedback

2.11. Registers view

This view displays registers on the selected device and allows users to edit the values of various registers. If a register contains bit fields, it can be expanded so that its bit fields can also be displayed. Register values that have changed are highlighted in yellow background. To manually change a register or register bit field value, single click in the value cell of the register and modify it. If the register is read-only, it will not allow you to modify it. The display format of the value of a register or bit field can be set to Default, Hexadecimal, Decimal, Binary, and Octal.

Following are the commands available in the view:

Image	Name	Description	Availability
📋	Copy Registers	Copies the selected registers to the system clipboard.	Context menu
⊟	Collapse All	Collapses all the expanded registers.	Toolbar action
	Number Format	Selects the display format for the values of individual items or all items in the view. The format at an individual items takes precedence over the format at the view.	To apply format on individual items, select the desired items and bring up context menu item Number Format to select a desired format. To apply format on all items in the view, select Toolbar View Menu and select Number Format menu item to select a desired format.
	Open New View	Opens a new view.	Toolbar action
🔄	Refresh	Refreshes the view.	Toolbar action
🔄	Continuous Refresh	Enables the view to automatically refresh on a regular basis.	Toolbar action
	Customize Continuous Refresh Interval	Sets the refresh interval for Continuous Refresh.	Toolbar View Menu action
	Select Columns	Selects to show or hide columns of the view.	Toolbar View Menu action and select Layout menu item and click Select Columns
	View Memory at Address	Shows the memory with the address of the selected register or bit field in a Memory View.	Context menu
	View Memory at Value	Shows the memory with the value of the selected register or bit field in a Memory View.	Context menu
X+Y =?	Watch	Create an expression for the selected register group, register, or bit field in Expressions View.	Context menu

See Also:
Real-time Debugging

Submit Code Composer Documentation Feedback

2.12. Introduction to target configuration

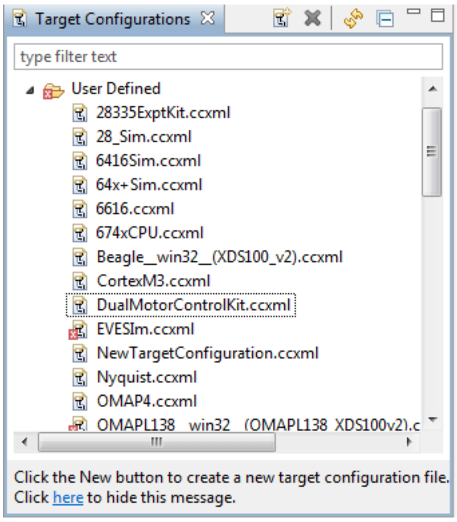
In migrating from the existing Code Composer, the target configuration selection method and organization have changed. These changes include:

1. Target configuration now uses a flexible XML schema that helps Texas Instruments and developers define the specific details. Instead of using the registry, Code Composer uses a hierarchy of related XML files to define a connection and board/device combination. Because the target configurations are XML files, they are easily deleted, copied, and displayed.

2. Separating the connection technique (XDS560, simulation, emulation) from the device/board selected. This implies:
   - You can change the emulation type a target configuration uses.
   - Third-party vendors do not need to create configurations that are device/board dependent. Every user chooses the connection type and the needed board.

4. Target Configurations are files that by default are stored in a user specific directory. However, target configuration files (files with .ccxml extension) can also be stored directly in projects to simplify the process of starting a debugger.

5. Target configurations are *automatically* added to Target Configurations View "User Defined" folder. Target Configurations view may be opened from View menu. A configuration can be designated as the `[Default]` configuration.



type filter text

- User Defined
  - 28335ExptKit.ccxml
  - 28_Sim.ccxml
  - 6416Sim.ccxml
  - 64x+Sim.ccxml
  - 6616.ccxml
  - 674xCPU.ccxml
  - Beagle__win32__(XDS100_v2).ccxml
  - CortexM3.ccxml
  - DualMotorControlKit.ccxml
  - EVESIm.ccxml
  - NewTargetConfiguration.ccxml
  - Nyquist.ccxml
  - OMAP4.ccxml
  - OMAPL138__win32__(OMAPL138_XDS100v2).c

Click the New button to create a new target configuration file.
Click here to hide this message.

6. Target configurations can be assigned to individual projects. These projects will automatically use the target configuration file that has an [Active] tag. If there is only one .ccxml file in a project then it autoamtically becomes active. When you create a new configuration, you can store them in the `Target Configurations` folder (by default) or browse to select the project/folder you want to assign this configuration.

Submit Code Composer Documentation Feedback

2.12.1. Target Configuration View

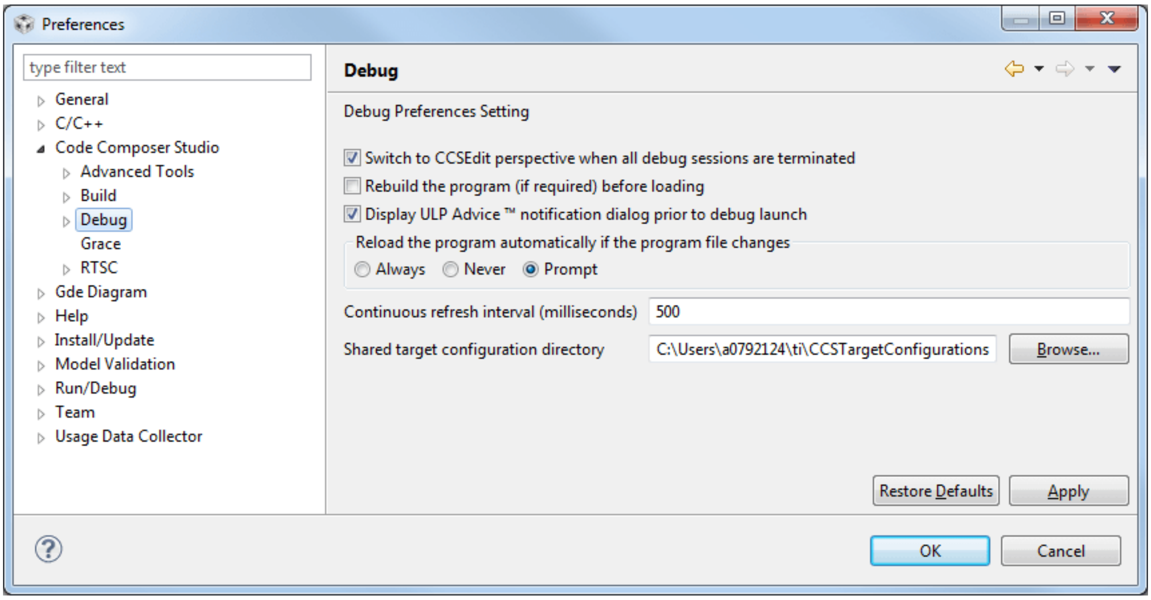
The Target Configuration View lets user view the target configuration files that are shipped with the product and manage user created target configurations. You can also create, edit, and remove target configuration files from this view.

These options are available.

Command	Name	Description	Availability
✚	New	Create a new target configuration file. You can add the configuration to an existing project or define the configuration as one of several under the User Defined section	View action
✖	Delete	Delete the selected target configuration file.	View action/Context menu
⟳	Refresh	refresh the view to pickup any target configuration resource change that are not detected by the view.	View action/Context menu
⊟	Collapse	collapse all nodes.	View action
	Configure	Set preferences for the memory view.	Context menu
	Launch Selected Configuration	launch the current selected configuration in the view.	Context menu
	Set As Default	set the selected configuration to be the default target configuration, when launch TI-Debugger is selected in the main window, this configuration will be use.	Context menu
	Properties	Show the file properties of the selected configuration file.	Context menu

Default Location For Target Configurations

All user-defined target configurations are kept in a default location. The default location can be changed on the **PreferencesC/C++** > **Debug** preference page.



Debug Preferences Setting

☑ Switch to CCSEdit perspective when all debug sessions are terminated

□ Rebuild the program (if required) before loading

☑ Display ULP Advice ™ notification dialog prior to debug launch

Reload the program automatically if the program file changes

○ Always ○ Never ● Prompt

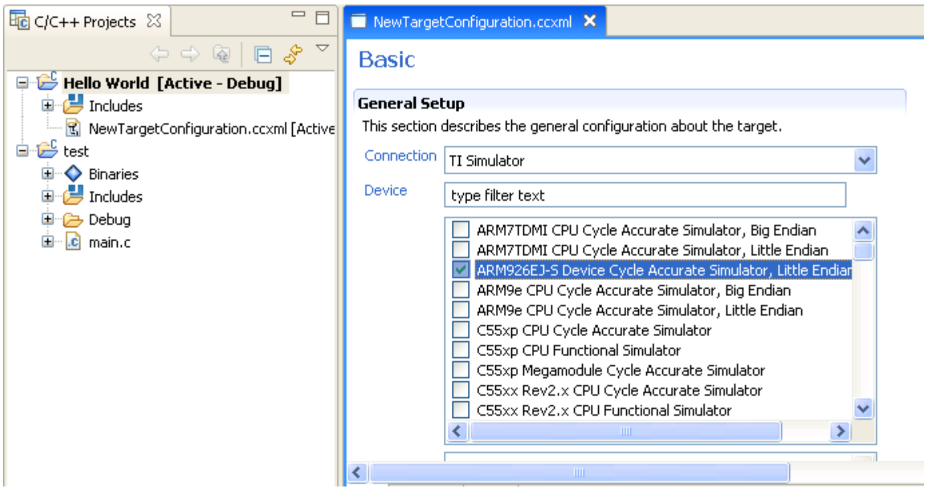
Continuous refresh interval (milliseconds): 500

Shared target configuration directory: C:\Users\a0792124\ti\CCSTargetConfigurations

2.12.2. General and Advanced Setup information

The **General Setup** area (**Basic** dialog tab) is used to:

- Construct a new target configuration
- Select a simple target configuration with a minimum of choices
- Update and modify target configurations


C/C++ Projects
Hello World [Active - Debug]
Includes
NewTargetConfiguration.ccxml [Active
test
Binaries
Includes
Debug
main.c
NewTargetConfiguration.ccxml
Basic

General Setup

This section describes the general configuration about the target.

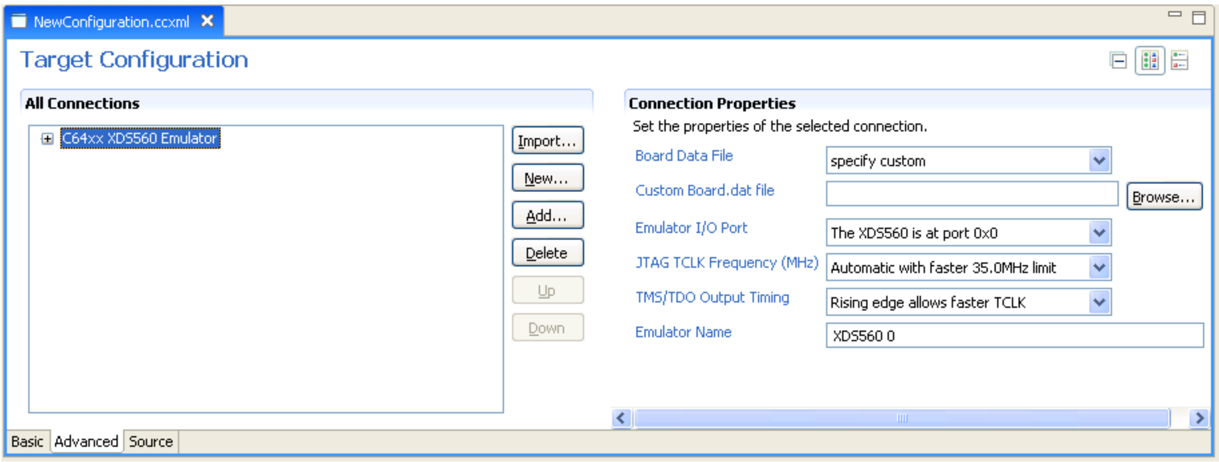
Connection: TI Simulator

Device: type filter text

□ ARM7TDMI CPU Cycle Accurate Simulator, Big Endian
□ ARM7TDMI CPU Cycle Accurate Simulator, Little Endian
☑ ARM926EJ-S Device Cycle Accurate Simulator, Little Endian
□ ARM9e CPU Cycle Accurate Simulator, Big Endian
□ ARM9e CPU Cycle Accurate Simulator, Little Endian
□ C55xp CPU Cycle Accurate Simulator
□ C55xp CPU Functional Simulator
□ C55xp Megamodule Cycle Accurate Simulator
□ C55xx Rev2.x CPU Cycle Accurate Simulator
□ C55xx Rev2.x CPU Functional Simulator

The **Advanced Setup Editor** (**Advanced** dialog tab) is used to:

- Modify one or more properties of an existing target configuration
- Customize your own unique target configuration with properties such as JTAG Clock Frequency, GEL initialization scripts, etc.
- Construct new target configurations by combining emulators, routers, and CPUs/boards.



Connection Properties

Set the properties of the selected connection.

Board Data File: specify custom

Custom Board.dat file:

Emulator I/O Port: The XDS560 is at port 0x0

JTAG TCLK Frequency (MHz): Automatic with faster 35.0MHz limit

TMS/TDO Output Timing: Rising edge allows faster TCLK

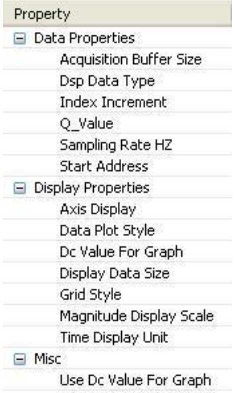
Emulator Name: XDS560 0

Submit Code Composer Documentation Feedback

2.13.1. Single Time

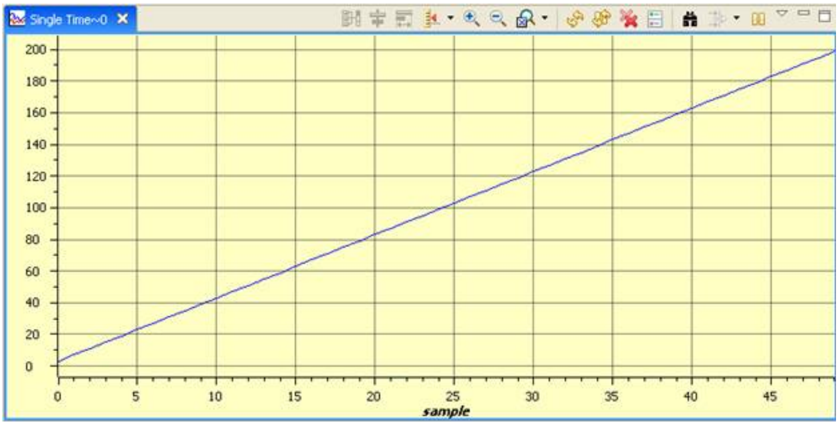
This graph plots a signal data versus time.

Applicable Properties



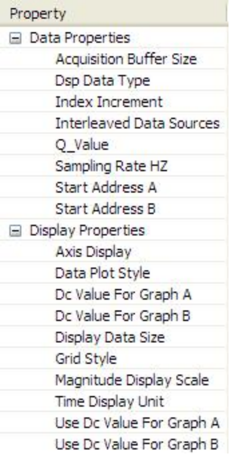
Property
⊟ Data Properties
Acquisition Buffer Size
Dsp Data Type
Index Increment
Q_Value
Sampling Rate HZ
Start Address
⊟ Display Properties
Axis Display
Data Plot Style
Dc Value For Graph
Display Data Size
Grid Style
Magnitude Display Scale
Time Display Unit
⊟ Misc
Use Dc Value For Graph

For a description please refer to Graph Properties Descriptions



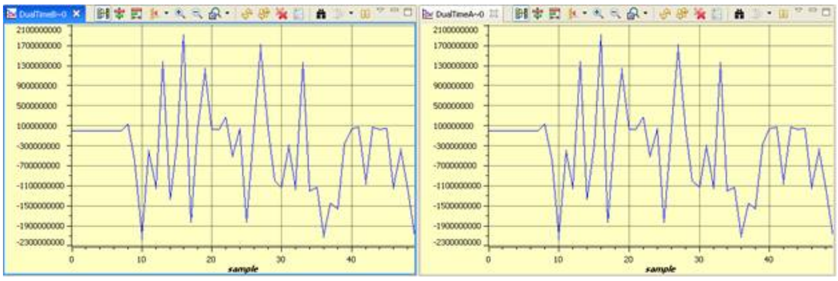
2.13.2. Dual Time

This graph plots the data in the display buffer on a magnitude versus time graph with no preprocessing. Two time traces of signal(s) are displayed on separated windows, allowing you to see the graphs side-by-side.



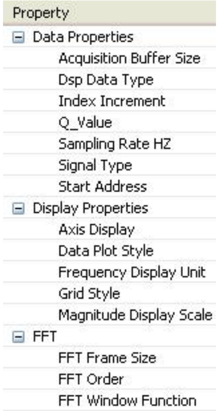
Property
⊟ Data Properties
Acquisition Buffer Size
Dsp Data Type
Index Increment
Interleaved Data Sources
Q_Value
Sampling Rate HZ
Start Address A
Start Address B
⊟ Display Properties
Axis Display
Data Plot Style
Dc Value For Graph A
Dc Value For Graph B
Display Data Size
Grid Style
Magnitude Display Scale
Time Display Unit
Use Dc Value For Graph A
Use Dc Value For Graph B

For a description please refer to Graph Properties Descriptions



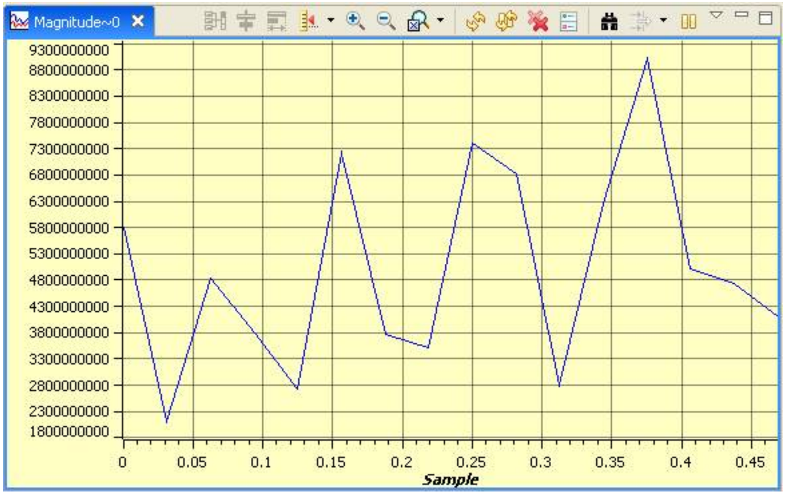
2.13.3. FFT Magnitude

Plots the magnitude part of an FFT transform FFT applied to a real or complex signal.



Property
⊟ Data Properties
Acquisition Buffer Size
Dsp Data Type
Index Increment
Q_Value
Sampling Rate HZ
Signal Type
Start Address
⊟ Display Properties
Axis Display
Data Plot Style
Frequency Display Unit
Grid Style
Magnitude Display Scale
⊟ FFT
FFT Frame Size
FFT Order
FFT Window Function

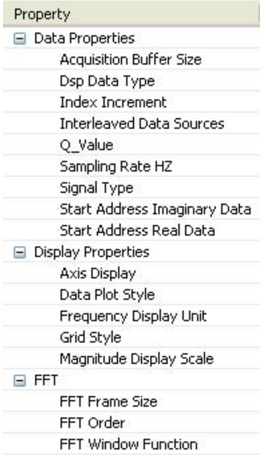
For a description please refer to Graph Properties Descriptions



2.13.4. FFT Magnitude & Phase

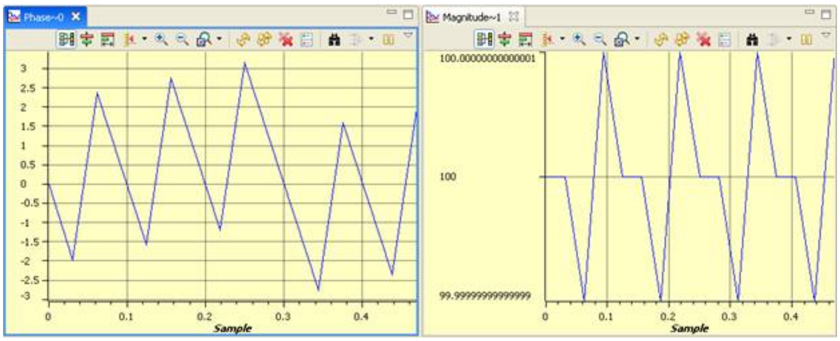
Plots both the Magnitude and phase of an FFT applied to a real or complex signal

Applicable Properties:



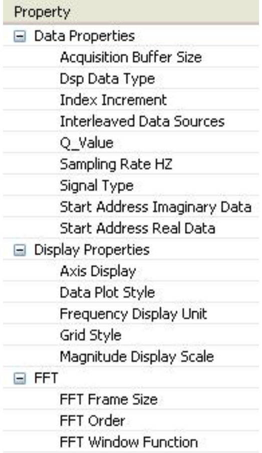
Property
⊟ Data Properties
Acquisition Buffer Size
Dsp Data Type
Index Increment
Interleaved Data Sources
Q_Value
Sampling Rate HZ
Signal Type
Start Address Imaginary Data
Start Address Real Data
⊟ Display Properties
Axis Display
Data Plot Style
Frequency Display Unit
Grid Style
Magnitude Display Scale
⊟ FFT
FFT Frame Size
FFT Order
FFT Window Function

For a description please refer to Graph Properties Descriptions



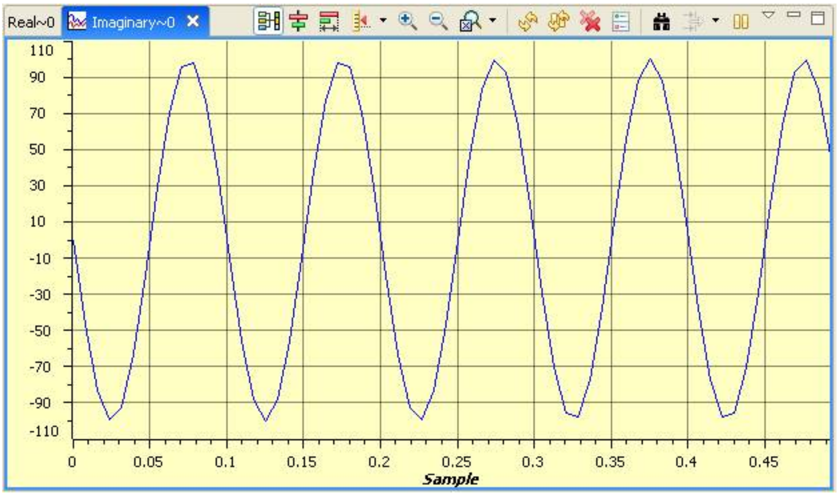
2.13.5. Complex FFT

The Complex FFT display consists of a real and imaginary data portion displayed on two graphs



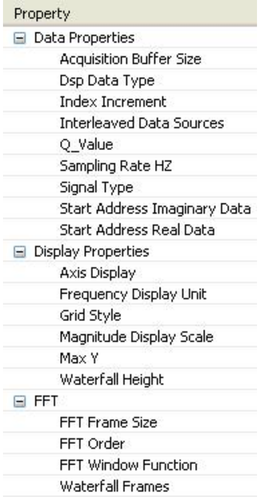
Property
⊟ Data Properties
Acquisition Buffer Size
Dsp Data Type
Index Increment
Interleaved Data Sources
Q_Value
Sampling Rate HZ
Signal Type
Start Address Imaginary Data
Start Address Real Data
⊟ Display Properties
Axis Display
Data Plot Style
Frequency Display Unit
Grid Style
Magnitude Display Scale
⊟ FFT
FFT Frame Size
FFT Order
FFT Window Function

For a description please refer to Graph Properties Descriptions



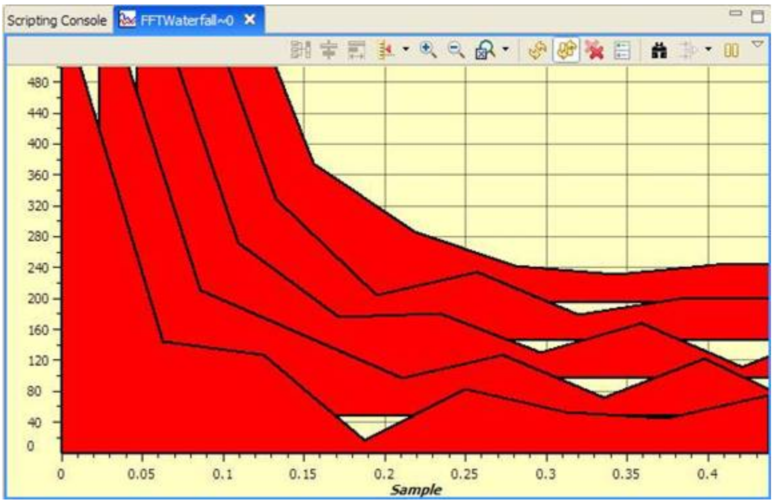
2.13.6. FFT Waterfall

FFT Waterfall Graph is used to observe FFT plot of a data location with the previous FFT plots for the same location whose values may have changed. Each plot will be of the form of solid shaded area under the cure, with older plots in the background and moving to the top and right of the Graph. The number of historical plots that will be shown can be chosen by user



Property
⊟ Data Properties
Acquisition Buffer Size
Dsp Data Type
Index Increment
Interleaved Data Sources
Q_Value
Sampling Rate HZ
Signal Type
Start Address Imaginary Data
Start Address Real Data
⊟ Display Properties
Axis Display
Frequency Display Unit
Grid Style
Magnitude Display Scale
Max Y
Waterfall Height
⊟ FFT
FFT Frame Size
FFT Order
FFT Window Function
Waterfall Frames

For a description please refer to Graph Properties Descriptions



2.13.7. Graph Properties

Each graph offers several properties that the user may set. The Table below shows what properties are available for each graph type:

Properties available for all Graphs

- **Acquisition Buffer Size**
- **DSP Data Type**
- **Index Increment**
- **Interleaved Data Sources**
- **Q Value**
- **Sampling Rate HZ**
- **Signal Type**
- **Start Address 1***
- **Grid Style**
- **Data Plot Style ****

*Called one of the following based on the graph and the signal Type (i.e. Real/Complex) and data interleaving

- **Start Address**
- **Start Address A**
- **Start Address Real Data**

** Not available to FFT Waterfall

For Graphs that can Display data from 2 sources (Dual Time, FFT Complex, FFT Magnitude Phase, FFT magnitude, FFT Waterfall) the following properties may (depending on the Signal Type) be made available.

- **Interleaved Data Sources**
- **Start Address 2 - Called one of the following based on the graph:**
  Start Address B
  Start Address Imaginary Data

Properties available for Time Graphs only

The following properties are available to time graphs only i.e. Single Time and Dual Time.

- **Display Data Size**
- **Time Display Unit**
- **Use DC value for Graph**
- **DC value for Graph**

Properties available for FFT Graphs only

The following properties are available to FFT graphs only i.e. FFT Magnitude, FFT Magnitude & Phase, FFT Complex, FFT Waterfall

- **Frequency Display Unit**
- **FFT Frame Size - Read-only property that is presented to the user; calculated based on the value of FFT order.**
- **FFT Order**
- **FFT Window Function**

Properties available for FFT Waterfall Graph only

- **MaxY**
- **Waterfall Height**
- **Waterfall Frames**

Property Descriptions

Property	Definition
Signal Type	Signal Type specifies the type of signal source to produce a particular graph. Two options are available for the Signal Type property: Real (corresponding to a single source display as in the case of a single time display)

	Complex (corresponding to two signal sources). When you select Complex, the Graph Property Dialog box displays the Interleaved Data Source option.
Interleaved Data Sources	Interleaved Data Sources specifies whether the signal sources are interleaved or not. Toggling this display option allows a single buffer input to represent two sources. Setting this option to Yes implies a 2-source input buffer, where the odd samples represent the first source and even samples represent the second. Setting this option to Yes creates the following additional field in the Graph Property Dialog box: Start Address is the starting location (on the actual/simulated target board) of the acquisition buffer containing the data to be graphed. When the graph is updated, the acquisition buffer, starting at this location, is fetched from the actual/simulated target board. The acquisition buffer then updates the display buffer, which is graphed. You can enter any valid C expression in the Start Address field. This expression is recalculated every time samples are read from the actual/simulated target. This means that if you enter a symbol in this field and the symbol later changes value, you do not have to re-enter this parameter. Setting Interleaved Data Sources to No creates the following additional fields: Start Address A Start Address B
Start Address	Start Address is the starting location (on the actual/simulated target board) of the acquisition buffer containing the data to be graphed. When the graph is updated, the acquisition buffer, starting at this location, is fetched from the actual/simulated target board. This acquisition buffer then updates the display buffer, which is graphed. You can enter any valid C expression in the Start Address field. This expression is recalculated every time samples are read from the actual/simulated target. This means that if you enter a symbol in this field and the symbol later changes value, you do not have to re-enter this parameter.
Acquisition Buffer Size	This is the size of the acquisition buffer you are using on your actual/simulated target board. For example, if you are processing samples one at a time, enter a 1 in this field. Enable the Left-Shifted Data Display field and connect the display to the correct location in your program. If your program processes an entire frame at one time (more than one sample) and you are only interested in that frame, enter the same value in the Acquisition Buffer Size and the Display Data Size fields. Then turn off the Left-Shifted Data Display option. When a graph is updated, the acquisition buffer is read from the actual/simulated target board and updates the display buffer. The display buffer is graphed. You can enter any valid C expression for the Acquisition Buffer Size field. This expression is recalculated every time samples are read from the actual/simulated target. Therefore, if you enter a symbol in this field and the symbol later changes values, you do not have to reenter this parameter.
Index Increment	The Index Increment field in the Time/Frequency Graph Properties dialog box allows you to specify the sample index increment for the graphical display of data. The value in this field is equivalent to a sample offset for non-interleaved sources. This offset permits you to extract signal data from multiple sources using a single graphical display. For example, an index increment of 2 corresponds to a sample offset value of 2, which, in turn, graphically displays every other sample in the acquisition buffer. You can therefore specify multiple data sources for display by entering a corresponding offset value in this field. This option provides a general specification for interleaved sources. If the Interleaved Data Sources is selected, the index increment option is disabled.
FFT Frame Size	FFT Framesize specifies the number of samples used in each FFT calculation. Note: The acquisition buffer can be a different size than the FFT Frame Size.
FFT Order	FFT Order specifies the FFT size = 2**FFT order
Waterfall Height	Number of Waterfall Frames specifies the number of waterfall frames to be displayed. Waterfall Height (%) specifies the percentage of vertical window height used to display a window frame.
Number of waterfall Frames	Number of Waterfall Frames specifies the number of waterfall frames to be displayed. Waterfall Height (%) specifies the percentage of vertical window height used to display a window frame.
FFT Windowing Function	FFT Windowing Function allows you to choose among the following windowing functions: Rectangle Bartlett Blackman Hanning Hamming These functions are performed on the data before the FFT calculation is performed.
Display Data Size	This is the size of the display buffer that you use. The contents of the display buffer are graphed on your screen. The display buffer resides on the host, so a history of your signal can be displayed even though it no longer exists on the actual/simulated target board. The size of the display is determined differently, depending on what you have selected for the time/frequency domain (Display Type) option. For a time domain graph, Display Data Size contains the number of samples that the graph displays. No preprocessing is done on the display buffer. Usually Display Data Size is greater than or the same as Acquisition Buffer Size. If Display Data Size is greater than Acquisition Buffer Size, the buffer data can be left shifted into the display buffer. For a frequency domain graph (FFT Magnitude, Complex FFT, FFT Magnitude and Phase), Display Data Size is represented by the FFT frame size (rounded up to the nearest power of 2) used for the FFT frequency analysis.

	You can enter any valid C expression for the Display Data Size field. This expression is recalculated every time samples are read from the actual/simulated target. Therefore, if you enter a symbol in this field which later changes values, you do not have to reenter this parameter
DSP Data Type	This field allows you to select among the following data types: 64-bit signed integer 64-bit floating point 32-bit signed integer 32-bit unsigned integer 32-bit floating point 16-bit signed integer 16-bit unsigned integer 8-bit signed integer 8-bit unsigned integer You can use signed integer in combination with the Q-Value field to interpret fixed-point values.
Q-Value	This field contains a nonzero Q-Value, which are fractional representations of integer values. The data on the actual/simulated target is interpreted using the Q-Value. They are formed by inserting a decimal space in the binary representation of an integer, resulting in greater precision. The Q-Value indicates amount of the displacement, according to the formula: New_integer_value = 2Q-Value A Q-Value of xx indicates a signed 2s complement integer whose decimal point is displaced xx places from the least significant bit (LSB).
Sampling Rate (Hz)	This field contains the sampling frequency for acquisition buffer samples, such as for analog to digital conversion. The sampling rate is used to calculate the time and frequency values displayed on the graph. For a time domain graph, this field calculates the values for the time axis. The axis is labeled from 0 to (Display Data Size * 1/Sampling Rate). For a frequency domain graph (FFT Magnitude, Complex FFT, FFT Magnitude and Phase), this field contains the number of samples (rounded down to the nearest power of 2) used for the FFT frequency analysis. The graph displays the frequency contents of the signal in the range from 0 to Sampling Rate/2
Axes Display	This option turns the X and Y axes in the graph window on and off. True False
Time Display Unit	This field Axes Display specifies the unit of measure for the time axis of the graph. This option and the value in the Sampling Rate field determine the values on the axis. You can select among the following values: s: second ms: millisecond us: microsecond sample: displays values in terms of the display buffer index
Frequency Display Unit	Frequency Display Unit specifies the unit of measure for the frequency axis of the graph. Select among the values: Hz kHz MHz
Magnitude Display Scale	This field sets the scaling function used for data values in the graph. You can choose between the following options: Linear - Plots the data on the linear scale Logarithmic Plots the data on the Log scale
Data Plot Style	This field sets how the data is visually represented in the graph. You can choose between the following options: Line - Connects data values linearly Bar Uses vertical lines to display values
Grid Style	This field sets the pattern of horizontal and vertical background lines in the graph. You can choose among the following options: No Grid Major Minor
DC Value for Graph	DC value is the Y-value which will be all ways remain midway of the Y-axis when the graph is plotted. For Graphs with 2 plots there will be 2 entries one for each

Use DC Value for Graph	Enable use of DC value. If not using DC value the Graph Y-Axis range will be based on the min and max value of the graph being plotted. For Graphs with 2 plots there will be 2 entries one for each

2.14. Profiling Code Execution

Performance is a key issue for embedded systems developers. As programs continue to grow in size and complexity, it becomes increasingly difficult for developers to isolate the subtle problems that can cause poor performance.

Profiling helps reduce the time it takes to identify and eliminate performance bottlenecks. The profiler analyzes program execution and shows where your program is spending its time. For example, a profile analysis can report how many cycles a particular function takes to execute and how often it is called.

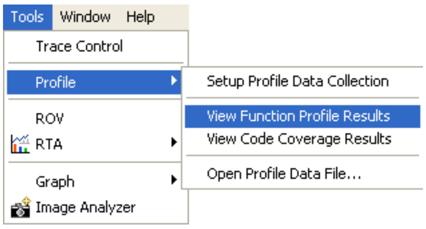
Profiling helps you to direct valuable development time toward optimizing the sections of code that most dramatically affect program performance.

Note: Compiler optimizations should be turned off or set to the lowest levels for accurate profiling data.

Profiling Features

- Using menus to access functionality
- Functional profiling
  - Function profiling view
- Code Coverage
  - Simulation based Code Coverage
    - Line coverage view
    - Function coverage view
  - Code Gen based Code Coverage
    - Line coverage view
    - Function coverage view
  - Line coloring in the source editor
- Scripting Profiling
- Profiling functions written in assembly

Accessing Profiling and Code Coverage


Tools
Window
Help
Trace Control
Profile
ROV
RTA
Graph
Image Analyzer
Setup Profile Data Collection
View Function Profile Results
View Code Coverage Results
Open Profile Data File...

Function Profiling

There are 3 steps to generate and visualize Functional profiling data:

- Step 1: Setup profiler to capture functional profiling data
- Step 2: Run .out file
- Step 3: Launch Profile View

Step 1: Setup profiler to capture functional profiling data

Open the **Profile Setup** view via *Tools* -> *Profile* -> *Setup Profile Data Collection* menu. The profile setup view is context sensitive to the debug session, so a debug session needs to be available and selected for the view to display the setup data. First we activate the view via the **Activate** button and then turn on function profiling by checking off the **Profile all Functions for CPU Cycles** box.

Step 2: Run .out file

Once the function profiling is configured, we can proceed to load and run a program. The function profile data should be visible as soon as the PCDT data is output via the simulator. The program does not need to terminate.

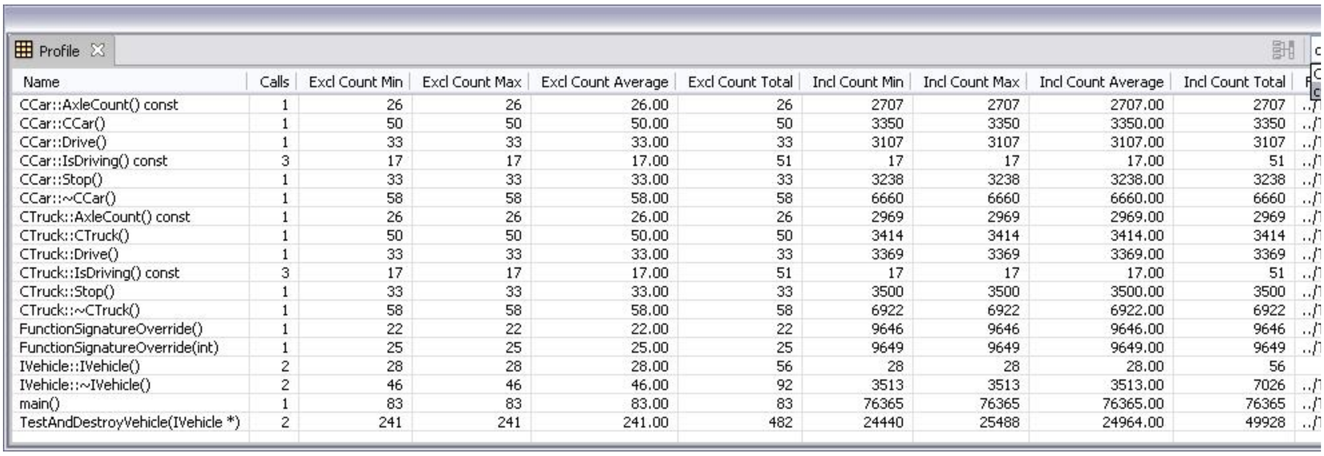
Step 3: Launch Profile View

We can visualize the data in the Functional Profiling view via *Tools* -> *Profile* -> *View Function Profile Results*. The Functional Profiling view is described in detail in the following section.

Functional Profiling View

Profiling results include function name, function call count, plus exclusive and inclusive counts (minimum, maximum, average, and total).
A drop-down can be selected to choose function profiling results for other event types that were included in Profile Setup.
Multiple function profile views can be opened to display each event type.
A menu option can be turned on/off to include TI library functions in the results.
The toolbar also contains filtering and find tools.

Results can be saved to a comma separated value (CSV) file, by selecting Data->Export in the view’s context menu.
The data can be displayed again with Tools->Profile->Open Profile Data File, or imported into a standard spreadsheet and data analysis software package.



Profile

Name	Calls	Excl Count Min	Excl Count Max	Excl Count Average	Excl Count Total	Incl Count Min	Incl Count Max	Incl Count Average	Incl Count Total	F
CCar::AxleCount() const	1	26	26	26.00	26	2707	2707	2707.00	2707	../
CCar::CCar()	1	50	50	50.00	50	3350	3350	3350.00	3350	../
CCar::Drive()	1	33	33	33.00	33	3107	3107	3107.00	3107	../
CCar::IsDriving() const	3	17	17	17.00	51	17	17	17.00	51	../
CCar::Stop()	1	33	33	33.00	33	3238	3238	3238.00	3238	../
CCar::~CCar()	1	58	58	58.00	58	6660	6660	6660.00	6660	../
CTruck::AxleCount() const	1	26	26	26.00	26	2969	2969	2969.00	2969	../
CTruck::CTruck()	1	50	50	50.00	50	3414	3414	3414.00	3414	../
CTruck::Drive()	1	33	33	33.00	33	3369	3369	3369.00	3369	../
CTruck::IsDriving() const	3	17	17	17.00	51	17	17	17.00	51	../
CTruck::Stop()	1	33	33	33.00	33	3500	3500	3500.00	3500	../
CTruck::~CTruck()	1	58	58	58.00	58	6922	6922	6922.00	6922	../
FunctionSignatureOverride()	1	22	22	22.00	22	9646	9646	9646.00	9646	../
FunctionSignatureOverride(int)	1	25	25	25.00	25	9649	9649	9649.00	9649	../
IVehicle::IVehicle()	2	28	28	28.00	56	28	28	28.00	56	
IVehicle::~IVehicle()	2	46	46	46.00	92	3513	3513	3513.00	7026	../
main()	1	83	83	83.00	83	76365	76365	76365.00	76365	../
TestAndDestroyVehicle(IVehicle *)	2	241	241	241.00	482	24440	25488	24964.00	49928	../

Code Coverage

There are 2 code coverage features available: Simulation based code coverage and Code gen based code coverage.

Simulation based Code Coverage

Simulation based code coverage data is derived from the ABA_PROG data output by the simulator.

There are 3 steps to generate and visualize Simulation based Code Coverage data:

- Step 1: Setup profiler to capture code coverage data
- Step 2: Run .out file to completion or CPU reset
- Step 3: Launch Code Coverage Views

Step 1: Setup profiler to capture code coverage data

Open the **Profile Setup** view via *Tools -> Profile -> Setup Profile Data Collection* menu. The profile setup view is context sensitive to the debug session, so a debug session needs to be available and selected for the view to display the setup data. First we activate the view via the **Activate** button and then turn on code coverage by checking off the **Collect Code Coverage and Exclusive Profile Data** box.

Step 2: Run .out file to completion or CPU reset

Once the profiler is configured, we can proceed to load and run a program. The code coverage data will not be visible until the program terminates or the CPU is reset. If the program does not terminate, you can halt the program and then reset the CPU to generate the data.

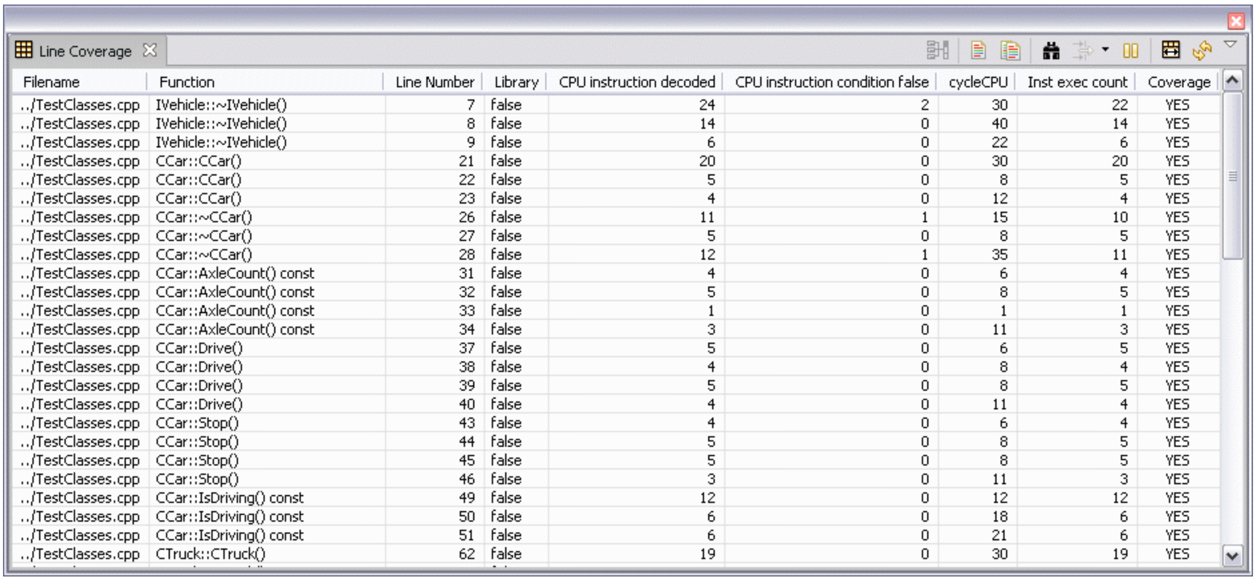
Step 3: Launch Code Coverage View

We can visualize the data in the Line Coverage and Function Coverage views via *Tools -> Profile -> View Code Coverage Results*. The Line Coverage and Function Coverage views are described in detail in the following two sections.

Line Coverage view

Columns include file name, function name, line number and coverage (full, partial or none). Line coverage can be shown in the source code, by selecting "Show Coverage". Selecting "Merge", merges code coverage results for multiple runs. When "Merge" is turned on, selecting "Clear" sets the results to be cleared on the next run. A menu option can be turned on/off to include TI library functions in the results. The toolbar also contains filtering and find tools.

Results can be saved to a comma-separated value (CSV) file, by selecting Data->Export in the view's context menu. The data can be displayed again with Tools->Profile->Open Profile Data File, or imported into a standard spreadsheet and data analysis software package.



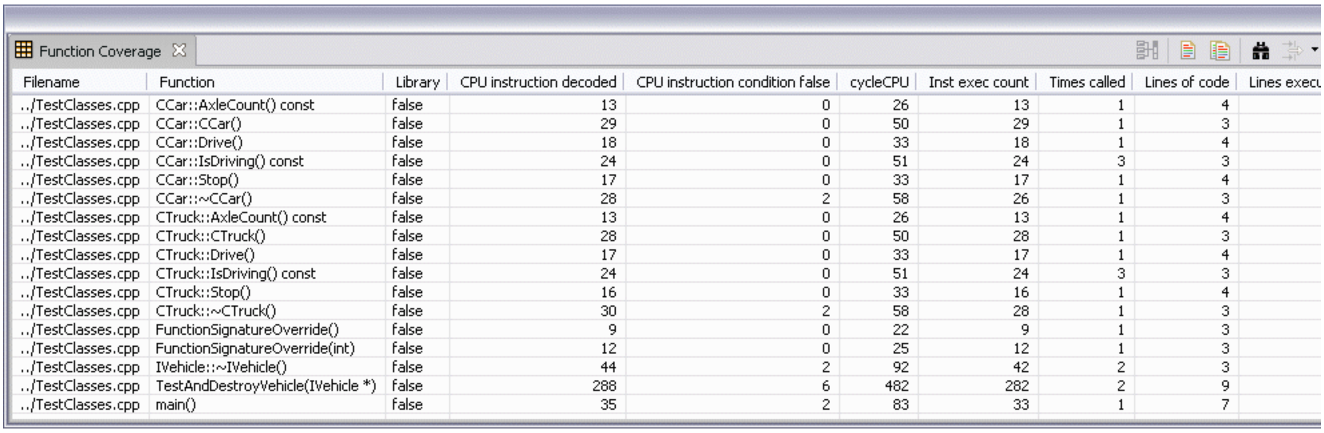
Line Coverage

Filename	Function	Line Number	Library	CPU instruction decoded	CPU instruction condition false	cycleCPU	Inst exec count	Coverage
../TestClasses.cpp	IVehicle::~IVehicle()	7	false	24	2	30	22	YES
../TestClasses.cpp	IVehicle::~IVehicle()	8	false	14	0	40	14	YES
../TestClasses.cpp	IVehicle::~IVehicle()	9	false	6	0	22	6	YES
../TestClasses.cpp	CCar::CCar()	21	false	20	0	30	20	YES
../TestClasses.cpp	CCar::CCar()	22	false	5	0	8	5	YES
../TestClasses.cpp	CCar::CCar()	23	false	4	0	12	4	YES
../TestClasses.cpp	CCar::~CCar()	26	false	11	1	15	10	YES
../TestClasses.cpp	CCar::~CCar()	27	false	5	0	8	5	YES
../TestClasses.cpp	CCar::~CCar()	28	false	12	1	35	11	YES
../TestClasses.cpp	CCar::AxleCount() const	31	false	4	0	6	4	YES
../TestClasses.cpp	CCar::AxleCount() const	32	false	5	0	8	5	YES
../TestClasses.cpp	CCar::AxleCount() const	33	false	1	0	1	1	YES
../TestClasses.cpp	CCar::AxleCount() const	34	false	3	0	11	3	YES
../TestClasses.cpp	CCar::Drive()	37	false	5	0	6	5	YES
../TestClasses.cpp	CCar::Drive()	38	false	4	0	8	4	YES
../TestClasses.cpp	CCar::Drive()	39	false	5	0	8	5	YES
../TestClasses.cpp	CCar::Drive()	40	false	4	0	11	4	YES
../TestClasses.cpp	CCar::Stop()	43	false	4	0	6	4	YES
../TestClasses.cpp	CCar::Stop()	44	false	5	0	8	5	YES
../TestClasses.cpp	CCar::Stop()	45	false	5	0	8	5	YES
../TestClasses.cpp	CCar::Stop()	46	false	3	0	11	3	YES
../TestClasses.cpp	CCar::IsDriving() const	49	false	12	0	12	12	YES
../TestClasses.cpp	CCar::IsDriving() const	50	false	6	0	18	6	YES
../TestClasses.cpp	CCar::IsDriving() const	51	false	6	0	21	6	YES
../TestClasses.cpp	CTruck::CTruck()	62	false	19	0	30	19	YES

Function Coverage view

Columns include file name, function name, line number and coverage (% of function covered) Line coverage can be shown in the source code, by selecting "Show Coverage". Selecting "Merge", merges code coverage results for multiple runs. When "Merge" is turned on, selecting "Clear" will set the results to be cleared on the next run. A menu option can be turned on/off to include TI library functions in the results. The toolbar also contains filtering and find tools.

Results can be saved to a comma separated value (CSV) file, by selecting Data->Export in the view's context menu. The data can be displayed again with Tools->Profile->Open Profile Data File, or imported into a standard spreadsheet and data analysis software package.



Function Coverage

Filename	Function	Library	CPU instruction decoded	CPU instruction condition false	cycleCPU	Inst exec count	Times called	Lines of code	Lines execu
../TestClasses.cpp	CCar::AxleCount() const	false	13	0	26	13	1	4	
../TestClasses.cpp	CCar::CCar()	false	29	0	50	29	1	3	
../TestClasses.cpp	CCar::Drive()	false	18	0	33	18	1	4	
../TestClasses.cpp	CCar::IsDriving() const	false	24	0	51	24	3	3	
../TestClasses.cpp	CCar::Stop()	false	17	0	33	17	1	4	
../TestClasses.cpp	CCar::~CCar()	false	28	2	58	26	1	3	
../TestClasses.cpp	CTruck::AxleCount() const	false	13	0	26	13	1	4	
../TestClasses.cpp	CTruck::CTruck()	false	28	0	50	28	1	3	
../TestClasses.cpp	CTruck::Drive()	false	17	0	33	17	1	4	
../TestClasses.cpp	CTruck::IsDriving() const	false	24	0	51	24	3	3	
../TestClasses.cpp	CTruck::Stop()	false	16	0	33	16	1	4	
../TestClasses.cpp	CTruck::~CTruck()	false	30	2	58	28	1	3	
../TestClasses.cpp	FunctionSignatureOverride()	false	9	0	22	9	1	3	
../TestClasses.cpp	FunctionSignatureOverride(int)	false	12	0	25	12	1	3	
../TestClasses.cpp	IVehicle::~IVehicle()	false	44	2	92	42	2	3	
../TestClasses.cpp	TestAndDestroyVehicle(IVehicle *)	false	288	6	482	282	2	9	
../TestClasses.cpp	main()	false	35	2	83	33	1	7	

Code Gen Based Code Coverage

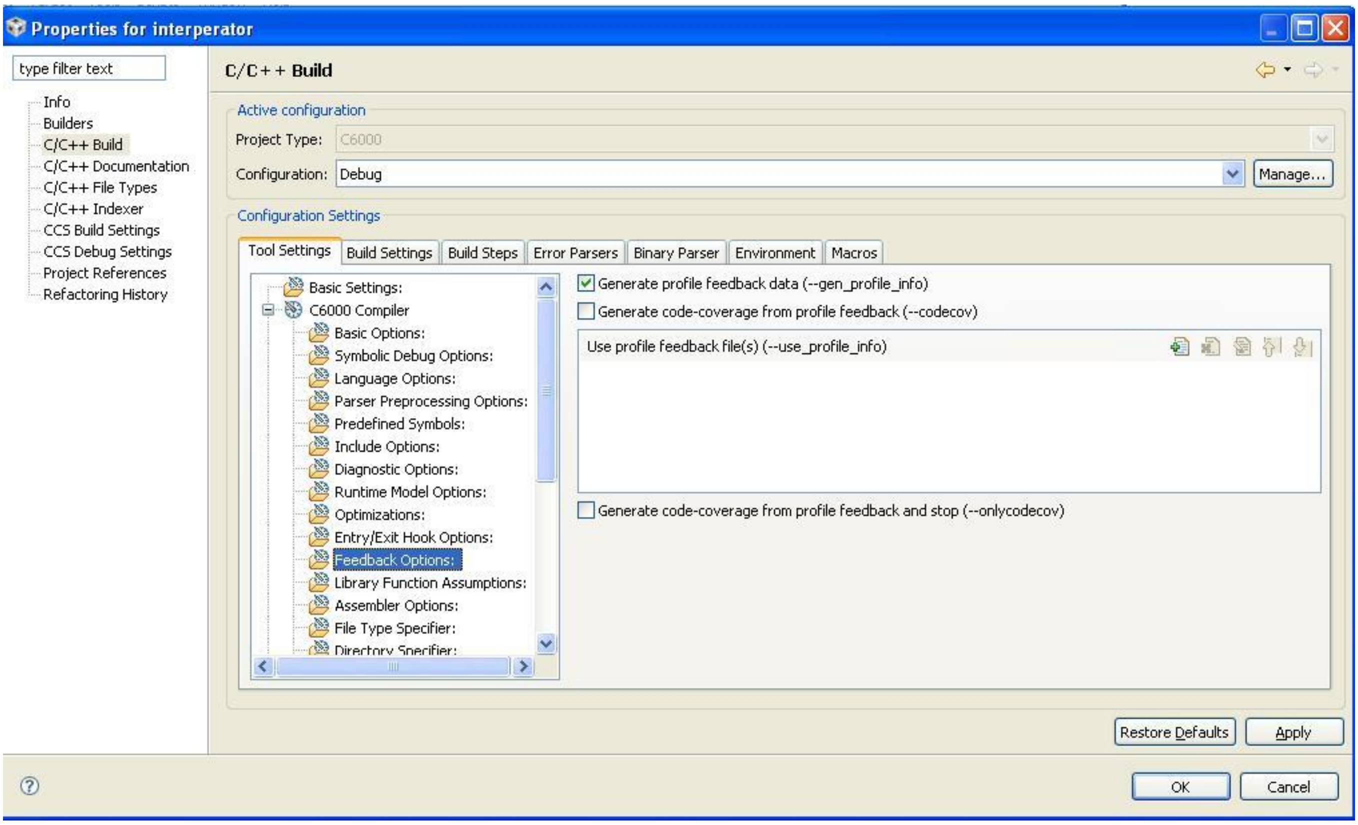
Code Gen based code coverage is coverage data derived from path profiling information collected by the C6000 compiler. Path-profiling identifies critical paths of a program by measuring the programs behavior at run time. This path profiling information can also be used to generate code coverage information. The CodeGen based code coverage feature will utilize this technique to generate compiler based code coverage information and visualize it within DVT.

There are 3 steps to generate CodeGen Based Code Coverage data:

- Step 1: Compile CCSV4 project with -gen_profile_info flag
- Step 2: Run .out file to generate .pdat raw data
- Step 3: Launch CodeGen feature

Step 1: Compile CCSV4 project with -gen_profile_info flag

To generate the profiling information the CCSV4 project must first be compiled with the **-gen_profile_info** flag. This flag can be set by right clicking on the project and selecting **properties**. The flag is seen on the **C/C++ Build -> Tool Settings -> Feedback Options:** view within the properties dialogue. Make sure the **Generate profile feedback data** option is checked and then recompile the project to generate an instrumented executable (.out file).



Properties for interperator

C/C++ Build

Active configuration

Project Type: C6000

Configuration: Debug

Configuration Settings

Tool Settings > C6000 Compiler > Feedback Options:

☑ Generate profile feedback data (--gen_profile_info)

□ Generate code-coverage from profile feedback (--codecov)

Use profile feedback file(s) (--use_profile_info)

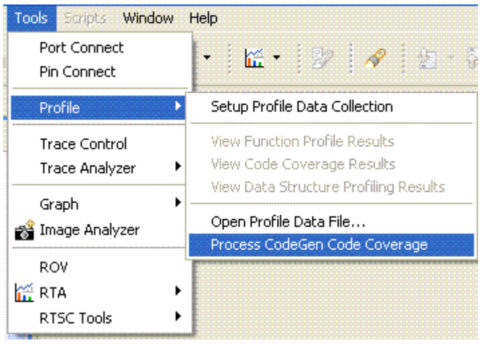
□ Generate code-coverage from profile feedback and stop (--onlycodecov)

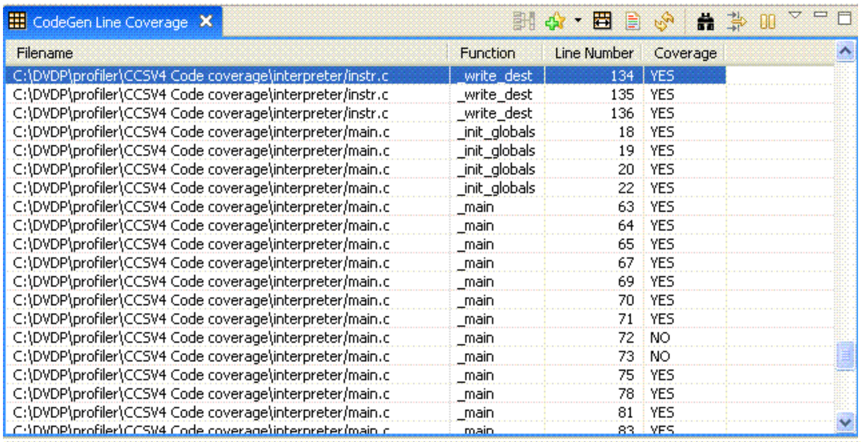
Step 2: Run .out file to generate .pdat raw data

The next step is to load the instrumented .out file on a target and run the program. This shall generate a **pprofout.pdat** file within the working directory (Debug) of the project.

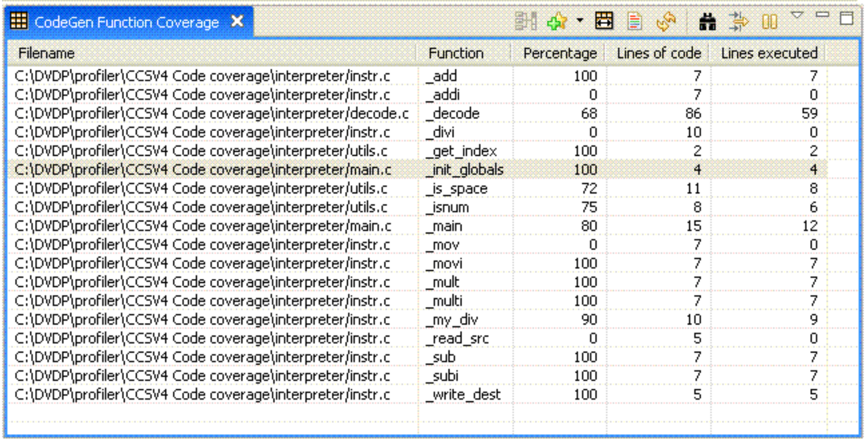
Step 3: Launch CodeGen feature

The last step is to make sure your project is the **Active project** and then run the codegen feature via the **Tools->Profile->Process CodeGen Code Coverage** menu. This shall launch the CodeGen Line Coverage and CodeGen Function Coverage views. These views are similar to the Line Coverage and Function Coverage views described here but contain CodeGen code coverage data.


Tools
Scripts
Window
Help
Port Connect
Pin Connect
Profile
Trace Control
Trace Analyzer
Graph
Image Analyzer
ROV
RTA
RTSC Tools
Setup Profile Data Collection
View Function Profile Results
View Code Coverage Results
View Data Structure Profiling Results
Open Profile Data File...
Process CodeGen Code Coverage



Filename	Function	Line Number	Coverage
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/instr.c	_write_dest	134	YES
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/instr.c	_write_dest	135	YES
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/instr.c	_write_dest	136	YES
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/main.c	_init_globals	18	YES
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/main.c	_init_globals	19	YES
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/main.c	_init_globals	20	YES
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/main.c	_init_globals	22	YES
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/main.c	_main	63	YES
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/main.c	_main	64	YES
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/main.c	_main	65	YES
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/main.c	_main	67	YES
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/main.c	_main	69	YES
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/main.c	_main	70	YES
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/main.c	_main	71	YES
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/main.c	_main	72	NO
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/main.c	_main	73	NO
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/main.c	_main	75	YES
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/main.c	_main	78	YES
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/main.c	_main	81	YES
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/main.c	_main	83	YES

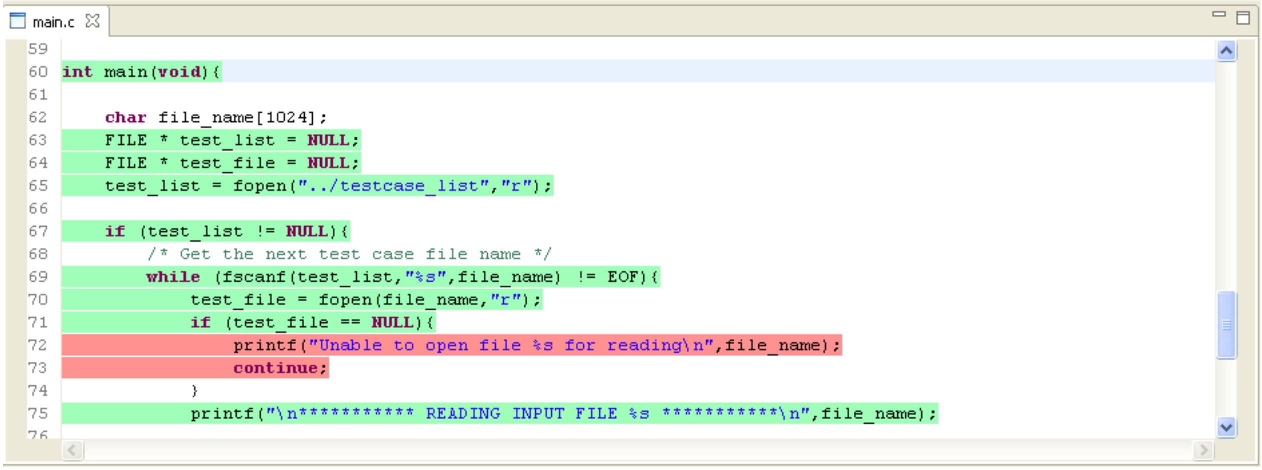


Filename	Function	Percentage	Lines of code	Lines executed
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/instr.c	_add	100	7	7
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/instr.c	_addi	0	7	0
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/decode.c	_decode	68	86	59
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/instr.c	_divi	0	10	0
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/utils.c	_get_index	100	2	2
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/main.c	_init_globals	100	4	4
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/utils.c	_is_space	72	11	8
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/utils.c	_isnum	75	8	6
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/main.c	_main	80	15	12
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/instr.c	_mov	0	7	0
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/instr.c	_movi	100	7	7
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/instr.c	_mult	100	7	7
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/instr.c	_multi	100	7	7
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/instr.c	_my_div	90	10	9
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/instr.c	_read_src	0	5	0
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/instr.c	_sub	100	7	7
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/instr.c	_subi	100	7	7
C:\DVDP\profiler\CCSV4 Code coverage\interpreter/instr.c	_write_dest	100	5	5

Line Coloring in Editor

The code coverage views have a tool bar action which updates file editor to indicate which lines of code were covered.

Green highlighting represents full line coverage.
Red highlighting represents no line coverage.



```
int main(void){

    char file_name[1024];
    FILE * test_list = NULL;
    FILE * test_file = NULL;
    test_list = fopen("../testcase_list","r");

    if (test_list != NULL){
        /* Get the next test case file name */
        while (fscanf(test_list,"%s",file_name) != EOF){
            test_file = fopen(file_name,"r");
            if (test_file == NULL){
                printf("Unable to open file %s for reading\n",file_name);
                continue;
            }
            printf("\n*********** READING INPUT FILE %s ***********\n",file_name);
```

Scripting Profiling

Profiling can be scripted, and results placed in .csv file(s).
The following is a JavaScript excerpt example for scripting function profiling.
The full example can be found in .\CCSv4\dvt\scripting\examples\javascript_examples\FunctionProfileScript.js
NOTE: profileAnalysisSession needs to be used before changing status of profileActivity to false.

Similarly, code coverage scripting can be done by changing the following in the script:

- **Replacing “Collect Code Coverage and Exclusive Profile Data” instead of “Profile all Functions for CPU Cycles” in the myDebugSession.profileSetup.getActivity() call.**
- **Replacing “CoverageProfile” in place of “FunctionProfile” in the dvtServer.openProfileAnalysisSession() call.**

```
. . .
            //Get the DVT Server
            var dvtServer = script.getServer("DVTServer.1");
            var profileAnalysisSession = dvtServer.openProfileAnalysisSession(myDebugSession, "FunctionProfile");
            var profileActivity = myDebugSession.profileSetup.getActivity("Profile all Functions for CPU Cycles");
            profileActivity.setStatus(true);
            try
            {
                        myDebugSession.target.run();
            }
            catch(e)
            {
                        //Timeout expected.
            }
            myDebugSession.target.halt();
            profileAnalysisSession.waitUntilProfilingComplete();
            var exports = profileAnalysisSession.exportData();
            for (var i=0; i < exports.length; i++)
            {
                        exports[i].save(exports[i].getName() + ".csv");
            }
. . .
```

Profiling functions written in assembly

For functions written in assembly, user should use the proper compiler supported assembly directives to declare and mark the function boundaries. This will give more meaningful data in the profiling results for such functions.

as an example for c6000 compiler version 7.x if the target code being built is of format ELF, compiler documentation says to use the .asmfunc and .endasmfunc directives for this purpose.

_load: **.asmfunc**

mv a4, b0
...
b b3
nop 5

.endasmfunc

Submit Code Composer Documentation Feedback

2.15.1. Getting Started

To get started load one of the examples provided here.

1. **Create an Image Analyzer View**

   Choose the menu Tools -> Image Analyzer from the main CCS toolbar. An Image view will be created and the Properties view opened.

2. **Import Image Properties**

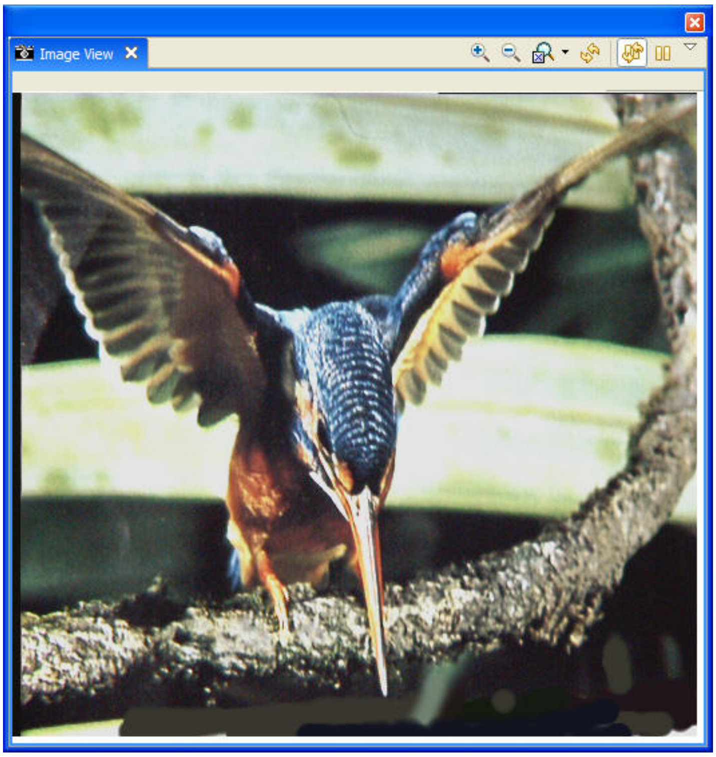
   In the Image view, right-click and select "Import Properties..." from the context menu

   Specify the file that contains the image properties you want to import (e.g. yuv420planar_file\yuv420planar_file_prop.txt)

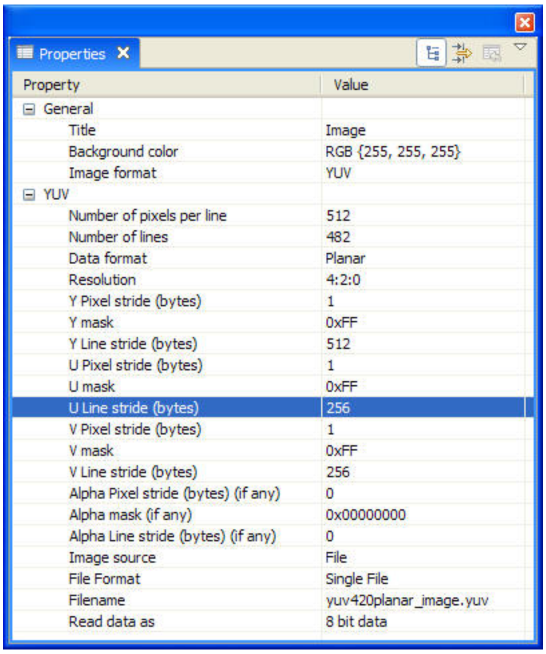
3. **Display Image**

   In the image view, click the Refresh button in the toolbar

   Image Analyzer fetches data from the file and processes the data. During this process, the progress is shown at the progress bar at the bottom of the main window



The Image properties are shown in the Properties view (the Image view must be selected for the Image properties to be shown):



Property	Value
⊟ General	
Title	Image
Background color	RGB {255, 255, 255}
Image format	YUV
⊟ YUV	
Number of pixels per line	512
Number of lines	482
Data format	Planar
Resolution	4:2:0
Y Pixel stride (bytes)	1
Y mask	0xFF
Y Line stride (bytes)	512
U Pixel stride (bytes)	1
U mask	0xFF
U Line stride (bytes)	256
V Pixel stride (bytes)	1
V mask	0xFF
V Line stride (bytes)	256
Alpha Pixel stride (bytes) (if any)	0
Alpha mask (if any)	0x00000000
Alpha Line stride (bytes) (if any)	0
Image source	File
File Format	Single File
Filename	yuv420planar_image.yuv
Read data as	8 bit data

2.15.2. Image Analyzer View

The Image view provides the following toolbar functions:

Zoom In

Zoom Out

Zoom Options

- **Reset**
- **Horizontal Zoom**
- **Vertical Zoom**
- **Both Directions**
- **Zoom Factor**

Refresh: Reloads the image data from the file or connected device

Refresh on Halt: If enabled the image is refreshed on a CPU halt

The Image view provides the following context menu functions:

Refresh: Reloads the image data from the file or connected device

Properties: Opens the Properties view and shows the Image properties

Import Properties: Import a file containing image properties

Export Properties: Export image properties to a file

Red Component: Display the red component only

Green Component: Display the green component only

Blue Component: Display the blue component only

RGB: Display all components

The Image view provides a cross hair function that displays the pixel values in RGB and the image's original format in the status bar of the view. To enable it move the mouse cursor over the image and hold the left button pressed.

2.15.3. Image Properties

General

- **Title**
- **Background Color**
- **Image Format: RGB, Bayer, YUV, Bitonal**

RGB

- **Number of pixels per line**
- **Number of lines**
- **Data format**
- **Pixel stride (bytes)**
- **Red mask**
- **Green mask**
- **Blue mask**
- **Alpha mask (if any)**
- **Line stride (bytes)**

Bayer

- **Number of pixels per line**
- **Number of lines**
- **Line order**
- **Pixel stride (bytes)**
- **Component mask**
- **Line stride (bytes)**

YUV

- **Number of pixels per line**
- **Number of lines**
- **Data format: Planar or Packed**
- **Resolution: 4:4:4, 4:2:2, 4:2:0, or 4:1:1**
- **YUV order: for Packed formats**
- **Y Pixel stride (bytes): The number of bytes to skip to get the value of the Y component for the next pixel. Typical values:**
  o Planar : 1 byte
  o Packed 4:4:4 : 3 bytes
  o Packed 4:2:2 or 4:2:0 : 2 bytes
- **Y mask: Image Analyzer reads packets of bytes of the size specified in Y Pixel stride. The byte order of the packets is Big Endian. Y mask specifies which of the bits in the packet represent the value of the Y component. Typical values:**
  o Planar : 0xFF
  o Packed 4:4:4 : 0xFF0000
  o Packed 4:2:2 or 4:2:0 : 0xFF00
- **Y Line stride (bytes): Only used for Planar formats. The number of bytes to skip to get to the first pixel of the next line (counting from the first pixel of the current line, i.e. (Number of pixels per line) x (number of bytes per pixel). This is typically the same as the Number of pixels per line.**
- **U Pixel stride (bytes): The number of bytes to skip to get the value of the U component for the next pixel. Typical values:**
  o Planar : 1 byte
  o Packed 4:4:4 : 3 bytes
  o Packed 4:2:2 or 4:2:0 : 4 bytes
- **U mask: Image Analyzer reads packets of bytes of the size specified in U Pixel stride. The byte order of the packets is Big Endian. U mask specifies which of the bits in the packet represent the value of the U component. Typical values:**
  o Planar : 0xFF
  o Packed 4:4:4 : 0xFF0000
  o Packed 4:2:2 or 4:2:0 : 0xFF000000
- **U Line stride (bytes): Only used for Planar formats. The number of bytes to skip to get to the first pixel of the next line (counting from the first pixel of the current line, i.e. (Number of pixels per line) x (number of bytes per pixel). Typical values:**
  o 4:4:4 : Number of pixels per line
  o 4:2:2 or 4:2:0 : (Number of pixels per line) / 2
  o 4:1:1 : (Number of pixels per line) / 4
- **V Pixel stride (bytes): The number of bytes to skip to get the value of the V component for the next pixel. Typical values:**
  o Planar : 1 byte
  o Packed 4:4:4 : 3 bytes
  o Packed 4:2:2 or 4:2:0 : 4 bytes
- **V mask: Image Analyzer reads packets of bytes of the size specified in V Pixel stride. The byte order of the packets is Big Endian. V mask specifies which of the bits in the packet represent the value of the V component. Typical values:**
  o Planar : 0xFF
  o Packed 4:4:4 : 0xFF0000
  o Packed 4:2:2 or 4:2:0 : 0xFF000000
- **V Line stride (bytes): Only used for Planar formats. The number of bytes to skip to get to the first pixel of the next line (counting from the first pixel of the current line, i.e. (Number of pixels per line) x (number of bytes per pixel). Typical values:**
  o 4:4:4 : Number of pixels per line
  o 4:2:2 or 4:2:0 : (Number of pixels per line) / 2
  o 4:1:1 : (Number of pixels per line) / 4
- **Alpha Pixel stride (bytes): The number of bytes to skip to get the value of the Alpha component for the next pixel. This is optional. If your image doesn’t have any Alpha component set this to 0.**
- **Alpha mask: Image Analyzer reads packets of bytes of the size specified in Alpha Pixel stride. The byte order of the packets is Big Endian. Alpha mask specifies which of the bits in the packet represent the value of the Alpha component. This is optional. If your image doesn’t have any Alpha component set this to 0.**
- **Alpha Line stride (bytes): Only used for Planar formats. The number of bytes to skip to get to the first pixel of the next line (counting from the first pixel of the current line), i.e. (Number of pixels per line) x (number of bytes per pixel). This is typically the same as the Number of pixels per line. This is optional. If your image doesn’t have any Alpha component set this to 0.**
- **Line stride: Only used for Packed formats. The number of bytes to skip to get to the first pixel of the next line (counting from the first pixel of the current line), i.e. (Number of pixels per line) x (number of bytes per pixel). Typical values are Number of pixels per line) x (Y Pixel stride).**

Bitonal

- **Foreground color**
- **Number of pixels per line**
- **Number of lines**
- **Pixel order**

Source

- **Image Source: Connected Device, File**
- **Start address**
- **Filename**
- **Read data as**
- **File Format (YUV only): Single File, Single File with Custom Offsets, Multiple Files**
- **Y Offset (bytes)**
- **U Offset (bytes)**
- **V Offset (bytes)**
- **Alpha Offset (bytes)**
- **Y Filename**
- **U Filename**
- **V Filename**
- **Alpha Filename**

2.15.4. Tasks

Import Image Properties

1. **Import Image Properties**
In the Image view, right-click and select "Import Properties..." from the context menu
Specify the file that contains the image properties you want to import (e.g. yuv420planar_file\yuv420planar_file_prop.txt)

2. **Display Image**
In the image view, click the 🔄 Refresh button in the toolbar
Image Analyzer fetches data from the file and processes the data. During this process, the progress is shown at the progress bar at the bottom of the main window

View an Image on a Target CPU

1. **Launch the debugger for the target**

2. **Select the CPU in the Debug view**

3. **Connect to the CPU, load a program and perform any other steps**

4. **Create an Image Analyzer View**
Choose the menu Tools -> 📷 Image Analyzer from the main CCS toolbar. An Image view will be created and the Properties view opened.

5. **Set Image Properties**
Either import the image properties from a file or enter them in the properties view. The properties view can be opened at any time from the context menu of the image view

6. **Display Image**
In the created image view, click the 🔄 Refresh button in the local toolbar
Image Analyzer fetches data from the file and processes the data. During this process, the progress is shown at the progress bar at the bottom of the main window

View an Image in a File

1. **Create an Image Analyzer View**
Choose the menu Tools -> 📷 Image Analyzer from the main CCS toolbar. An Image view will be created and the Properties view opened.

2. **Set Image Properties**
Either import the image properties from a file or enter them in the properties view. The properties view can be opened at any time from the context menu of the image view

   In the Image Source property, select File. If applicable, in the Image Format property select whether the image data is contained in multiple files, in a single file or in a single file with custom offsets to the image components.

   Specify the filename(s) in Filename property

   If applicable specify the custom offsets

3. **Display Image**
In the image view, click the 🔄 Refresh button in the local toolbar
Image Analyzer fetches data from the file and processes the data. During this process, the progress is shown at the progress bar at the bottom of the main window

2.16. RTOS Object View (ROV)

RTOS Object View: Using ROV for Eclipse-Based Debugging

The RTOS Object View (ROV) is a debugging tool you can use with applications that make use of RTSC content in CCStudio. This viewer provides state information about all the RTSC modules in the application. ROV is a **stop-mode** debugging tool, which means it can receive data about an application only when the target is halted, not when it is running.

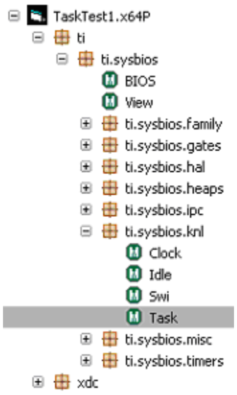
Contents

ROV Setup

Before using ROV in CCS 4, verify that the path to the XDC tools and your XDC package path have been setup properly. All of the packages used in the configuration of the application must be on the XDC package path for ROV to open.

If you are in a project debug session, ROV will get your paths from your project properties. If you are in a debug session that is not associated with a project, ROV will get your paths from the CCS 4 workspace settings.

Opening ROV



To open ROV, follow these steps:

1. **Load your application for debugging. If you are using a multi-core device, select the device you want to debug before opening ROV.**
2. **Choose Tools->RTOS Object View (ROV) from the CCStudio menus. This opens the ROV area at the bottom of the CCStudio window. (You can open ROV at any time while you have an application loaded.)**
3. **Select a module from the expandable tree (see figure to left) of RTSC packages used by the application.**
4. **Select one of the tabs in the right pane of the ROV (see "ROV tabs" for details about the tabs).**
5. **With the application at a breakpoint, wait for data to be loaded.**

Viewing Data with ROV

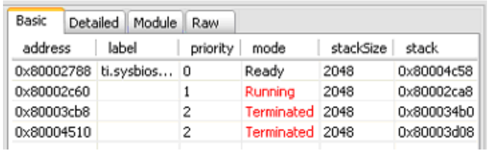
The left side of the ROV GUI displays the modules in the application, organized as a tree according to the package hierarchy. Double-clicking a module in the tree shows its state information on the right side.

When you click on a module or tab in ROV, you see the "Acquiring data..." message while ROV retrieves and processes information. If you switch to a different tab or module, ROV cancels the previous data request and begins a new one. Data for non-selected tabs is not loaded in the background when you switch tabs or modules.

Acquiring data...

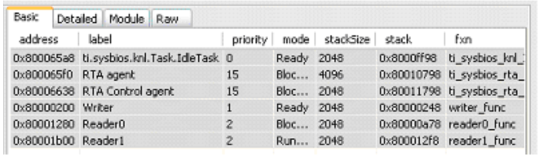
When you reload or rerun the application, ROV clears all of its cached data.

When the application halts at a breakpoint, ROV refreshes only the currently displayed information. If any data has changed since the last time ROV requested that particular data, ROV displays that data in **red text**. Keep in mind, however, that ROV only retrieves data when it is requested. If ROV did not get an item at the last breakpoint, it has nothing to compare it to at the current breakpoint. In short, if a field is not red, this does not necessarily mean that the data did not change.



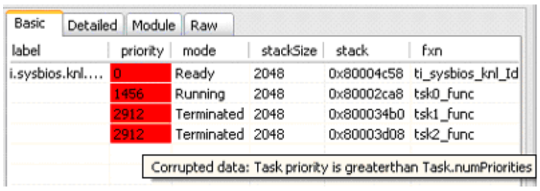
address	label	priority	mode	stackSize	stack
0x80002788	ti.sysbios...	0	Ready	2048	0x80004c58
0x80002c60		1	Running	2048	0x80002ca8
0x80003cb8		2	Terminated	2048	0x800034b0
0x80004510		2	Terminated	2048	0x80003d08

While the target is running, you may continue to click around ROV. Any data that was retrieved at the last breakpoint is shown in **gray text**. If data was not retrieved you see a "Target running..." message that indicates that ROV cannot retrieve new data while the target is running.



address	label	priority	mode	stackSize	stack	fxn
0x800065a8	ti.sysbios.knl.Task.IdleTask	0	Ready	2048	0x8000ff98	ti_sysbios_knl_
0x800065f0	RTA agent	15	Bloc...	4096	0x80010798	ti_sysbios_rta_
0x80006638	RTA Control agent	15	Bloc...	2048	0x80011798	ti_sysbios_rta_
0x80000200	Writer	1	Ready	2048	0x80000248	writer_func
0x80001280	Reader0	2	Bloc...	2048	0x80000a78	reader0_func
0x80001b00	Reader1	2	Run...	2048	0x800012f8	reader1_func

ROV highlights errors by making the background red . Hover your mouse over a field with a red background to see the error message.



label	priority	mode	stackSize	stack	fxn
i.sysbios.knl....	0	Ready	2048	0x80004c58	ti_sysbios_knl_Id
	1456	Running	2048	0x80002ca8	tsk0_func
	2912	Terminated	2048	0x800034b0	tsk1_func
	2912	Terminated	2048	0x80003d08	tsk2_func

Corrupted data: Task priority is greaterthan Task.numPriorities

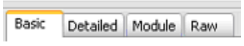
ROV can detect the following types of errors.

- **Validation errors. RTSC modules can provide validation of information ROV receives from the target. For example, a particular structure field can have a maximum value of 32. If the value on the target is 33, the module can report this as a problem.**
- **Memory. ROV may report bad memory accesses or other detectable problems.**
- **ROV errors. ROV reports standard exceptions if a corrupted data from the target causes ROV-related code in a module to behave incorrectly.**

ROV tabs

ROV can show state information for every module in an application that maintains a state.

Data in ROV is displayed in multiple tabs. Each tab provides a different type or level of data. The list of tabs depends on the module, but the most common tabs are "Basic", "Detailed", "Module", and "Raw".



For details about the data available in ROV for a particular module, refer to the online help for that module.

Basic and Detailed tabs

The "Basic" and "Detailed" tabs contain state information about the selected module's instances. Each row of the table represents an instance of the module. ROV tabs that contain instance information always have the instance's address (the address of the instance's state structure) in the first column.

The "Basic" and "Detailed" tabs contain largely the same information. The "Detailed" tab generally provides detailed information that may require longer processing time. If you do not need the extra information in the "Detailed" tab, you may use the "Basic" tab for somewhat quicker response times.

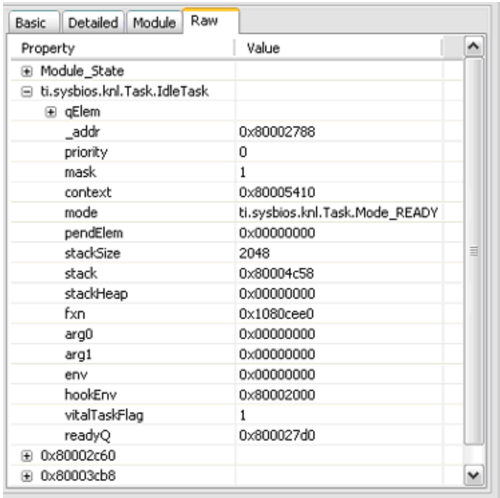
Module tab

The "Module" tab presents module state information. It has only one row, which represents the module's state object.

Raw tab

The list of tabs varies between modules, but all modules have a "Raw" tab. Other tabs are implemented by the module writer so that the raw data from the target has been filtered, processed, and formatted to be presentable. In the "Raw" tab, however, the raw data from the target is displayed.

The "Raw" tab has expandable items for the module's state structure and one for each module instance. Expanding these shows the contents of each state structure.



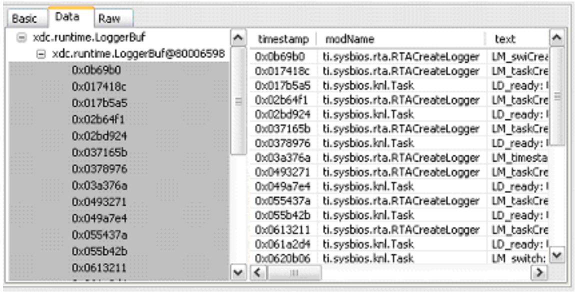
Property	Value
⊞ Module_State	
⊟ ti.sysbios.knl.Task.IdleTask	
⊞ qElem	
_addr	0x80002788
priority	0
mask	1
context	0x80005410
mode	ti.sysbios.knl.Task.Mode_READY
pendElem	0x00000000
stackSize	2048
stack	0x80004c58
stackHeap	0x00000000
fxn	0x1080cee0
arg0	0x00000000
arg1	0x00000000
env	0x00000000
hookEnv	0x80002000
vitalTaskFlag	1
readyQ	0x800027d0
⊞ 0x80002c60	
⊞ 0x80003cb8	

Data tab

Some modules have a "Data" tab. This tab displays tables of data associated with the instances managed by a module. A good example is a memory heap. The data tab for a memory heap module could show all the heap's allocable memory blocks.

A data tab may have a name other than "Data" to describe the data it displays, such as "Records" or "Blocks".

A "Data" tab lists the module's instances on the left. Expanding an instance shows a list of all of the data elements. Selecting one or more data elements on the left shows details about them on the right. You can select all the elements in an instance by clicking the first element, holding down the Shift key, and clicking the last element.



⊟ xdc.runtime.LoggerBuf
⊟ xdc.runtime.LoggerBuf@80006598
0x0b69b0
0x017418c
0x017b5a5
0x02b64f1
0x02bd924
0x037165b
0x0378976
0x03a376a
0x0493271
0x049a7e4
0x055437a
0x055b42b
0x0613211

timestamp	modName	text
0x0b69b0	ti.sysbios.rta.RTACreateLogger	LM_swiCre
0x017418c	ti.sysbios.rta.RTACreateLogger	LM_taskCre
0x017b5a5	ti.sysbios.knl.Task	LD_ready:
0x02b64f1	ti.sysbios.rta.RTACreateLogger	LM_taskCre
0x02bd924	ti.sysbios.knl.Task	LD_ready:
0x037165b	ti.sysbios.rta.RTACreateLogger	LM_taskCre
0x0378976	ti.sysbios.knl.Task	LD_ready:
0x03a376a	ti.sysbios.rta.RTACreateLogger	LM_timesta
0x0493271	ti.sysbios.rta.RTACreateLogger	LM_taskCre
0x049a7e4	ti.sysbios.knl.Task	LD_ready:
0x055437a	ti.sysbios.rta.RTACreateLogger	LM_taskCre
0x055b42b	ti.sysbios.knl.Task	LD_ready:
0x0613211	ti.sysbios.rta.RTACreateLogger	LM_taskCre
0x061a2d4	ti.sysbios.knl.Task	LD_ready:
0x0620b06	ti.sysbios.knl.Task	LM_switch:

3.1. Workbench and Perspectives

The term Workbench refers to the desktop development environment. The Workbench aims to achieve seamless tool integration and controlled openness by providing a common paradigm for the creation, management, and navigation of workspace resources.

When the Workbench is launched the first thing you see is a dialog that allows you to select where the workspace should be located. The workspace is the directory where your work will be stored.

After the workspace location is chosen, a single Workbench window is displayed. A Workbench window offers one or more perspectives. A perspective contains editors and views, such as the Projects View. Multiple Workbench windows can be opened simultaneously. A perspective defines the initial set and layout of views in the Workbench window. A specific perspective may also customize menus/toolbars that are visible. This is to minimize the menu toolbar clutter and focus the user interface on the task at hand(e.g. debugging or editing). Each perspective shares the same set of editors. Each perspective provides a set of functionality aimed at accomplishing a specific type of task or works with specific types of resources.

For example, the Debug perspective contains the views that you would use while debugging C/C++ programs. As you work in the Workbench, you will probably switch perspectives frequently.

You can use the Window > Perspectives preference page to open perspectives in the same window or in a new window.

Perspectives

The primary perspectives used in Code Composer Studio are the CCS Edit and CCS Debug perspectives. Select the CCS Edit perspective to create or build C/C++ projects. By default it includes an editor area and the following views: Project Explorer (the file navigator for C/C++ resources) and Outline view. Errors in a project build will appear in the Console and Problems windows, which are opened automatically when a build is started.

CCS Debug perspective is automatically enabled when user starts a debugger. The debug perspective is used for debugging C/C++ projects. By default it includes an editor area and the following views: debug, opened source files, variables, expressions, and registers.

Window layout is automatically preserved as user makes changes.

Customize Perspective

User may adjust toolbars and menus that appear in a particular perspective by selecting Window->Customize perspective. Commands tab may be used to enable/disable groups of functionality. Functionality groups may add new menus and/or toolbars.

Reset Perspective

A particular perspective may be reset by selecting Window->Reset Perspective.

Views

- Debug View
- Breakpoint View
- Scripting Console
- Target Configuration View
- Disassembly View
- Pin Connect View
- Port Connect View
- Memory View
- Target Voltage View
- Profile Setup
- Cache View
- Variables View
- Expressions View
- Registers View
- **Trace View**

Submit Code Composer Documentation Feedback

3.2.1. Starting TI Debugger

CCS debugger may be started through a project or it may just use a target configuration file to start the debugger. If you have a project in your workspace, then you may add a target configuration file to your project, if you have only one .ccxml file then it should automatically become [Active]. Clicking on Debug toolbar button will start debugger for project that is currently selected. You may also any file belonging to a project be the active editor in focus to start a project debug action. CCS will automatically determine which proejct a file belongs to and start debugger for that project. Debug project command automatically connects the target and loads the program to that target.

Alternatively user may start a debugger by creating a target configuration and starting a debugger through Target Configuration View. Create new target configuration file "File -> New ->Target Configuration File" ; select connection and target, open target configuration view, select newly created target configuration file, right click on select Launch Debugger. Use this command if you just want to connect to a target quickly, with minimal or no initial steps required. For instance, you may want to view or change memory or registers on the target. Or you may want to connect to a running target and debug the execution of the program from there. In some multi-CPU systems, you have to first modify some registers on one CPU to allow debugging on another CPU. For all these cases, using this command helps you to start debugging a CPU without performing initial steps like download program, run until `main`, and so forth. This command starts Debug Sessions for all CPUs of the Target System.

Launch Configurations

Changes made in the project’s Debug Properties are associated with a specific project, but are kept in the *Launch Configurations*.

Launch Configurations track information for specific debug sessions, including these items:

- Target system
- CPU debugged
- Program to download
- Specific Debug properties for that CPU
- Location of the source files associated with that program

Launch Configurations are created any time you start a debugger.

You can see all Launch Configurations, change their properties, add new ones, or delete existing ones by select **Run > Debug Configurations...** in CCS Debug perspective. This dialog displays all the launch configurations:

Changing a Launch configuration from that dialog is equivalent to changing the Project’s debug properties. You can also start a Debug Session from the **Debug** dialog or from the **Launch Recent Programs** toolbar button and menus.

Submit Code Composer Documentation Feedback

3.2.3. Program Load and Program Reload

The COFF or ELF file (`*.out`) produced by building your program must be loaded onto the actual or simulated target board, prior to execution or debugging. Project based debug session(when a debugger is started by selecting a project), automatically loads a program. In project less debug session (when a debugger is started from Target Configurations view), user must explicitly load a program or symbols to perform source based debug.

Program code and data are downloaded onto the target at the addresses specified in the program file. Symbols are loaded into a symbol table maintained by the debugger on the host. The symbols are loaded at the code and data addresses specified in the program file.

A program file can be loaded by selecting **Run > Load > Load Program** , which automatically loads the COFF file for the active debug project.

Once you have halted the program, you can use the **Run > Load > Reload Program** command.

Loading Symbols Only

It is useful to load only symbol information when working in a debugging environment where the debugger cannot or need not load the object code, such as when the code is in ROM.

Symbols can be loaded by selecting **Run > Load > Load Symbols** command which loads the symbols from the active COFF file.

You can also use the **Run > Load > Add Symbols** to create additional symbols.

The debugger deletes any previously loaded symbols from the symbol table maintained on the host. The symbols in the symbol file are then loaded into the symbol table. Symbols are loaded at the code and data addresses specified in the symbol file. This command does not modify memory or set the program entry point.

You can also specify a code offset and a data offset that the debugger applies to each symbol in the specified symbol file. For example, if you have a symbol file for an executable that contains code addresses starting at `0x100` and data addresses starting at `0x1000`. However, in the program loaded on the target, the corresponding code starts at `0x500100` and the data is located at `0x501000`.

To specify the code and data offset, select the Modules view and click the Load Symbols icon in the Modules toolbar. A Load Symbols dialog opens, with options to enter a Code offset or Data offset address.

The debugger automatically offsets every symbol in that symbol file by the given value.

Adding Symbols Only

Symbol information can also be appended to the existing symbol table. This command differs from the Load Symbol command in that it does not clear the existing symbol table before loading the new symbols, but just appends the new symbols to the existing symbol table.

The steps for adding symbol information, with **Run > Load > Add Symbols**, is similar to the steps outlined for loading symbols.

Submit Code Composer Documentation Feedback

3.3.3. Configuring Projects

A project stores all the information needed to build an individual program or library, including:

- Filenames of source code and object libraries
- Build-tool options
- Include file dependencies
- Build-tool version used to build the project.

Code Composer helps you to configure your files and project options.

- Creating a Project
- Project Configurations
- Project Dependencies

Submit Code Composer Documentation Feedback

3.6.1. Adding a DSP target configuration

You can create a new target configuration (connection plus board/device variant) using several methods:

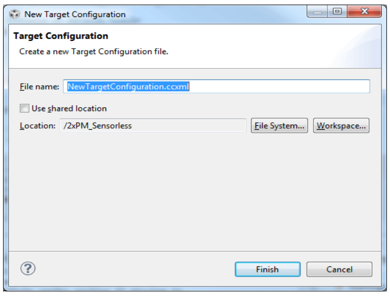
- Adding a new target configuration to an existing Code Composer project
- Adding a new target configuration to the list of target configurations

Adding to an existing Code Composer project

Once you have defined a project, you can add a configuration to this project using these commands:

- Select the project in Project Explorer and then select **File > New > Target Configuration** command from the main menu.
- Alternatively, select a project in the **C/C++ Projects** view, right-click, and select the **New > Target Configuration file** command.

1. Using one of the commands from the list, display the **New Target Configuration** dialog.



Target Configuration

Create a new Target Configuration file.

File name: NewTargetConfiguration.ccxml

☐ Use shared location

Location: /2xPM_Sensorless

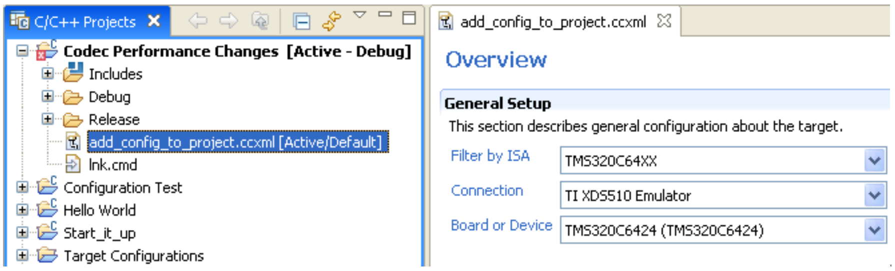
2. Type a file name and and click **Finish**.

4.

Note: When a new target configuration is assigned to a project, the target configuration becomes the **Active Target Configuration**. See Selecting default target and project configurations.)

5. Use the procedures in Selecting Target Configuration... to determine the connect and device variant for this configuration.

6. Select **File > Save** (💾) to record your target configuration selections.


C/C++ Projects
Codec Performance Changes [Active - Debug]
Includes
Debug
Release
add_config_to_project.ccxml [Active/Default]
lnk.cmd
Configuration Test
Hello World
Start_it_up
Target Configurations
add_config_to_project.ccxml
Overview

General Setup

This section describes general configuration about the target.

Filter by ISA: TMS320C64XX

Connection: TI XDS510 Emulator

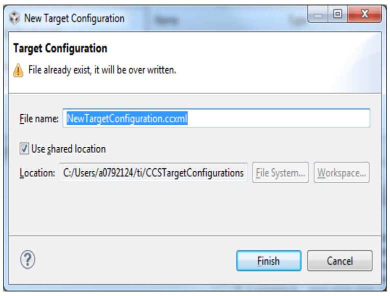
Board or Device: TMS320C6424 (TMS320C6424)

Note: You can add multiple target configurations to any of your projects. However only one target configuration is the *Default Target Configuration* and only one configuration is designated as the *Active Debug Configuration*. One target configuration can assume both roles, as illustrated in the previous graphic.

Top of Page

Adding a new target configuration to the list of target configurations

1. Using one of the commands from the list, display the **New Target Configuration** dialog.



Target Configuration

⚠ File already exist, it will be over written.

File name: NewTargetConfiguration.ccxml

☑ Use shared location

Location: C:/Users/a0792124/ti/CCSTargetConfigurations

-

Type a file name and click **Finish**.

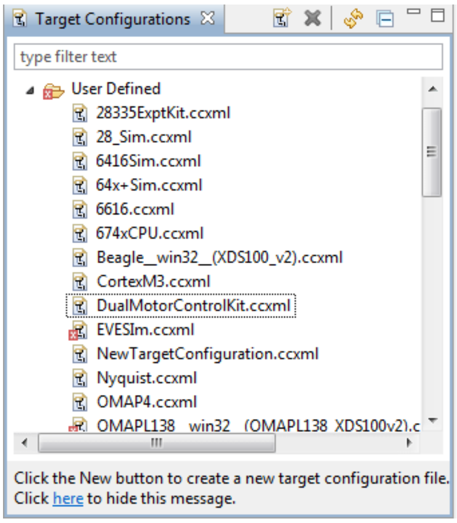
-
-

Use the procedures in Selecting Target Configuration... to determine the connection and device variant for this configuration.

-

Select **File > Save** (💾) to record your target configuration selections.

You may see your newly create target configuration from Target Configurations View (**View -> Target Configurations**) and use Target Configurations view to start project less debug sessions.



28335ExptKit.ccxml
28_Sim.ccxml
6416Sim.ccxml
64x+Sim.ccxml
6616.ccxml
674xCPU.ccxml
Beagle__win32__(XDS100_v2).ccxml
CortexM3.ccxml
DualMotorControlKit.ccxml
EVESIm.ccxml
NewTargetConfiguration.ccxml
Nyquist.ccxml
OMAP4.ccxml
OMAPL138__win32__(OMAPL138_XDS100v2).c

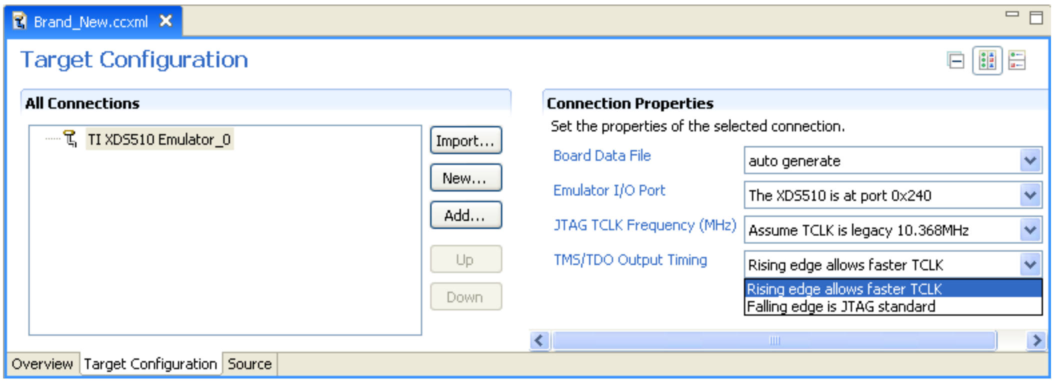
Click the New button to create a new target configuration file.
Click here to hide this message.

Top of Page

Submit Code Composer Documentation Feedback

3.6.2. Building a target configuration

To build a target configuration using a connection, boards, devices, and CPUs use the procedure in Selecting Configuration using target properties to display the initial Advanced Setup view. This display shows All Connections (and any existing components) and the Connection Properties available for your selected connection.



TI XDS510 Emulator_0

Set the properties of the selected connection.

Board Data File: auto generate
Emulator I/O Port: The XDS510 is at port 0x240
JTAG TCLK Frequency (MHz): Assume TCLK is legacy 10.368MHz
TMS/TDO Output Timing: Rising edge allows faster TCLK
Rising edge allows faster TCLK
Falling edge is JTAG standard

At this junction, you can:

- **Add** a component under the current connection
- Create a **New** connection as part of this target configuration
- **Import** an entire configuration
- Change the connection properties for the selected connection
- Change the connection priority using **Up** or **Down** buttons (multiple connections must exist)

Submit Code Composer Documentation Feedback

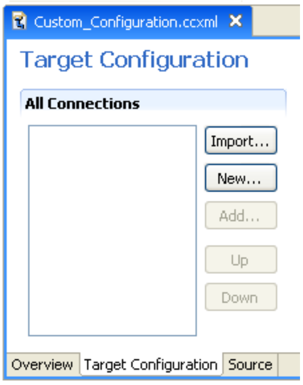
3.6.3. Creating a custom target configuration

To define a custom target configuration:

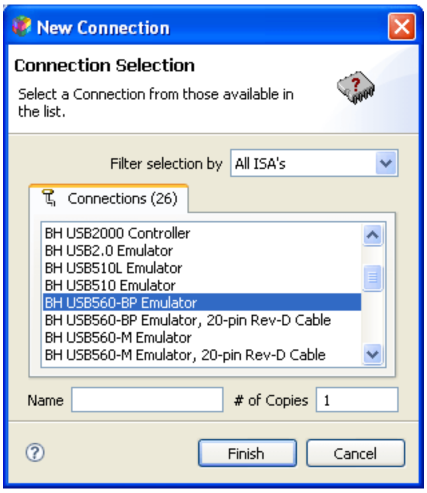
1. Select the **Target > New Target Configuration** command from the main menu.

2. Type a name for this target configuration in the **New Target Configuration** dialog. Click **Finish**.

3. In the Target Configuration <*name.ccxml*> view, select the **Target Configuration** dialog tab or the link by the **Advanced Settings**. The **All Connections** structure tree displays without any connections.



4. Click the **New** button. When the **New Connection** dialog displays, select a connection (for example, *BH USB560-BP Emulator*) or click **Browse** at the end of the **Connections** list to search for a connection XML file.



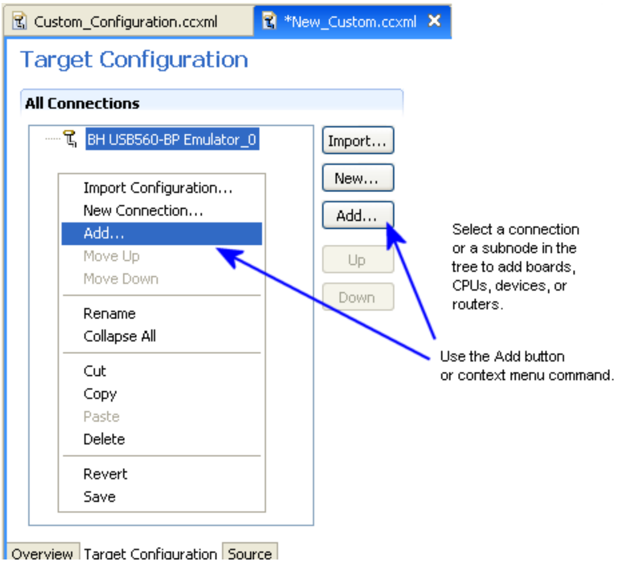
Select a Connection from those available in the list.

Filter selection by All ISA's

BH USB2000 Controller
BH USB2.0 Emulator
BH USB510L Emulator
BH USB510 Emulator
BH USB560-BP Emulator
BH USB560-BP Emulator, 20-pin Rev-D Cable
BH USB560-M Emulator
BH USB560-M Emulator, 20-pin Rev-D Cable

# of Copies 1

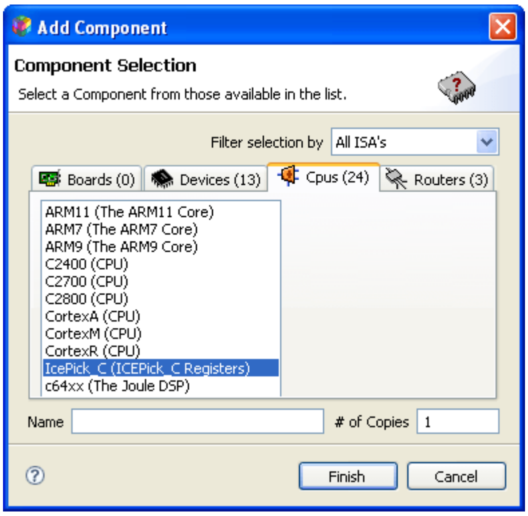
5. Click **Finish** in the **New Connection** dialog. Select the connection when it displays in the **All Connections** tree.



Select a connection or a subnode in the tree to add boards, CPUs, devices, or routers.

Use the Add button or context menu command.

6. When the **Add Component** dialog displays, select the desired category with the **Boards**, **Devices**, **Cpus**, or **Routers** dialog tab.



Add Component

Component Selection

Select a Component from those available in the list.

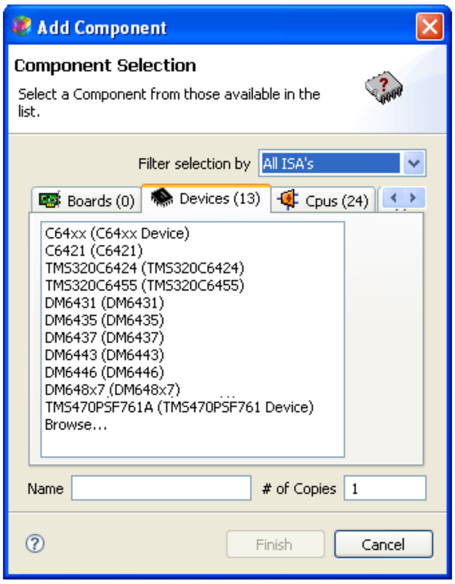
Filter selection by: All ISA's

Boards (0) | Devices (13) | Cpus (24) | Routers (3)

ARM11 (The ARM11 Core)
ARM7 (The ARM7 Core)
ARM9 (The ARM9 Core)
C2400 (CPU)
C2700 (CPU)
C2800 (CPU)
CortexA (CPU)
CortexM (CPU)
CortexR (CPU)
IcePick_C (ICEPick_C Registers)
c64xx (The Joule DSP)

Name: | # of Copies: 1

7. Select the desired component from the category list or select **Browse** to search for a component XML file.



Add Component

Component Selection

Select a Component from those available in the list.

Filter selection by: All ISA's

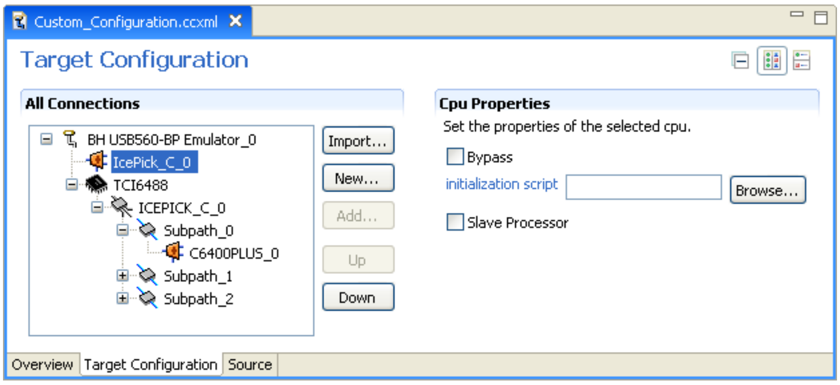
Boards (0) | Devices (13) | Cpus (24)

C64xx (C64xx Device)
C6421 (C6421)
TMS320C6424 (TMS320C6424)
TMS320C6455 (TMS320C6455)
DM6431 (DM6431)
DM6435 (DM6435)
DM6437 (DM6437)
DM6443 (DM6443)
DM6446 (DM6446)
DM648x7 (DM648x7)
TMS470PSF761A (TMS470PSF761 Device)
Browse...

Name: | # of Copies: 1

8. Click **Finish** to add the component to the All Connections tree.

9. Repeat steps six through eight to add another component. Use the **Up** and **Down** buttons to prioritize which component is debugged.



Custom_Configuration.ccxml

Target Configuration

All Connections

```
BH USB560-BP Emulator_0
    IcePick_C_0
    TCI6488
        ICEPICK_C_0
            Subpath_0
                C6400PLUS_0
            Subpath_1
            Subpath_2
```

Cpu Properties

Set the properties of the selected cpu.

□ Bypass

initialization script: | Browse...

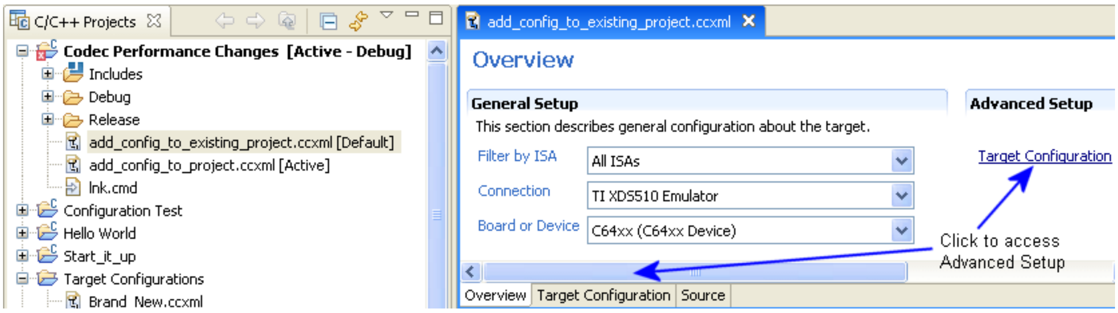
□ Slave Processor

10. Use the **File > Save** (💾) command to save the Target Configuration selections.

Submit Code Composer Documentation Feedback

3.6.4. Selecting Configuration using target properties

Once you have named a new target configuration and added the configuration to an existing project or defined the configuration as one of several under the Target Configurations section, you need to select the specific *connection* and *device/board* that define this *unique* connection.



This section describes general configuration about the target.

Filter by ISA: All ISAs

Connection: TI XDS510 Emulator

Board or Device: C64xx (C64xx Device)

In the [Selecting Target Configuration - connections, boards, and devices](#) you used the **General Setup** area to quickly select a desired connection/device combination. In the **Advanced Setup** you can use the more detailed configuration hierarchy and change CPU properties.

To access the **Advanced Setup** area with an existing configuration:

1. Click the **Target Configuration** dialog tab (bottom of view) or the link under the **Advanced Setup**.

To access the **Advanced Setup** area to [build a new configuration](#):

1. Select a connection under the **General Setup**.

2. Click the **Target Configuration** dialog tab (bottom of view) or the link under the **Advanced Setup**.

3. Click the **Add** button.

See [Building a Target Configuration](#) or [Importing a Target Configuration](#).

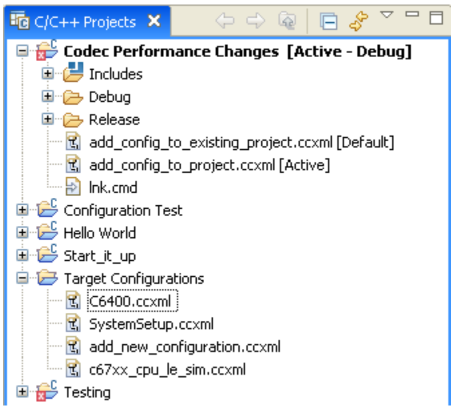
[Submit Code Composer Documentation Feedback](#)

3.6.5. Selecting default target and project configurations

You can assign a **Default Target Configuration** designation to a target configuration attached to a:

- Project
- **Target Configurations** folder

You can assign the **Active Target Configuration** designation to a *single* project. The project name displays in bold font.



To assign a project the Active Target Configuration or Default Target Configuration:

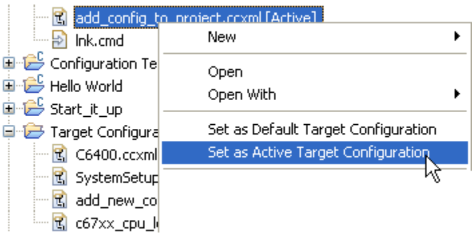
1. Select *a project* with MB1.

2. Right-click and select the **Set as Default Target Configuration** or **Set as Active Target Configuration** command.

To assign a Target configuration as the Default Target Configuration:

1. Select a *configuration* under the **Target Configurations** folder with MB1.

2. Right-click and select the **Set as Default Target Configuration** command.



Note: When a new target configuration is assigned to a project, the target configuration becomes the **Active Target Configuration**.

[Submit Code Composer Documentation Feedback](#)

3.6.6. Configuring TI Debugger for Existing Target Configuration

Once you have named a new target configuration and [added the configuration to an existing project](#) or [defined the configuration as one of several under the Target Configurations](#) section, you need to select the specific *connection* and *device/board* that define this *unique* connection.

You may adjust debugger settings during a debug session. Start a debugger and then click on Tools -> Debugger Options -> Generic Debugger Options to adjust settings. Depending on the device there may also be device specific debug options.

If you are working a project then you may also adjust debugger options through your project. Right click on a project and choose Properties, then navigate to Debug category. In order for Debug properties to be visible a project must be cofnigured for a specific device. see target configuration topics on how to create a target configuration and associate it with your project.

Submit Code Composer Documentation Feedback

3.7. Uninstalling CCS

Uninstall CCS

- To completely uninstall CCS you will need to exit CCS and navigate to CCSInstallRoot/ccsv6/ directory and double click on uninstall_ccs executable. Alternatively, on Windows you may go to Add/Remove program..

Submit Code Composer Documentation Feedback

3.8. The General Extension Language (GEL)

The General Extension Language (GEL) is an interpretive language similar to C that lets you create functions to extend Code Composer Studio IDE's usefulness. You create your GEL functions using the GEL grammar and then load them into the IDE. With GEL, you can access actual/simulated target memory locations and add options to the IDE�s GEL menu. GEL is particularly useful for automated testing and user workspace customization. You can call GEL functions from anywhere that you can enter an expression. You can also add GEL functions to the Watch window so they execute at every breakpoint.

Related Topics

GEL Grammar

Built-In GEL Functions

GEL Macros

Loading and Unloading GEL Files

Adding GEL Functions to the Menu Bar Using Keywords

3.8.1. GEL Grammar

GEL is a subset of the C programming language. GEL supports the following types of statements:

GEL Function Definition

GEL Functions: Alphabetical List

GEL Function Parameters

Calling GEL Functions

Local variables for functions

Function recursion

return statement

if and if-else statements

for statement

while and do-while statements

break statement

File I/O objects

Hardware breakpoints

GEL comments

Preprocessing statements

The following code shows how a GEL function is defined. Once a function is loaded, you can execute the function anytime by calling it with the correct Parameters. Calls such as: MyFunc(100, 0) or MyFunc(200) are both valid.

```
MyFunc(Parameters1, Parameters2)
{
if (Parameters1 == Parameters2)
a = Parameters1;
else
{
b = Parameters2;
c--;
}
}
```

The symbols a, b, and c are assumed to be target variables.

3.8.1.1. GEL Function Definition

GEL functions are defined as follows, where items in courier font are variables:

```
funcName( [Parameters1 [, Parameters2 ... [, Parameters6 ] ] ] )

{
statements
}
```

funcNameGEL function

Parameters Valid GEL Parameters

statements Valid GEL statements

GEL functions are defined in text files with a .gel extension. A GEL file can contain many GEL function definitions.

GEL functions do not identify any return type or need any header information to define the types of Parameters they require. This information is obtained automatically from the data value.

As with standard C, a GEL function definition cannot be embedded within another GEL function definition.

In this case, we are squaring the value the user passes to the function.

```
square(a)
{
return a*a;
}
```

If you add a call to this function in the Watch window, it looks like this:

```
square(1.2) = 1.44
square(5) = 25
```

Since a is a GEL Parameters, you do not have to define it in the function or in the target code.

You can follow each Parameters with an optional string that describes the use of the Parameters. This description is used in the dialog box that is created for dialog functions. For example:

```
dialog Init(filename "File to be Loaded",
cpuName "CPU Name",
initValue "Initialization Value")
{
GEL_Load(filename, cpuName);
a = initValue;
}
```

The dialog adds this function to the menu bar. Strings are given for the Parameters to provide a description on the Parameters entry dialog box. In the statement a = initValue, the letter a is not defined in the Parameters list; therefore, it must be defined on the actual/simulated target. If it is not, an error occurs when you call this function. Note the call to the built-in function GEL_Load; this function requires a string identifying the file name for the first Parameters and the CPU name. The CPU name Parameters is optional and is useful in setting up multiple processors. You must pass a string for the first Parameters. An example of a valid call to this function is:

```
Init("c:\\mydir\\myfile.out", "cpu_a", 0)
```

3.8.1.2. Local Variables for Functions

Within GEL functions, you can declare local variables and you can use variables defined in your target program. When a target variable is evaluated, the Code Composer Studio debugger obtains the necessary information from the target. The COFF file containing the symbol information must already be loaded.

In the following example, the variable i is declared within the GEL function and the variable data is assumed to be a target symbol.

```
MyFunc(num)
{
int i;
for (i=0; i<=num; i++)
data[i] = 0;
}
```

It is possible to use globals within GEL functions (See Global Variables for GEL Functions).

3.8.1.3. Global Variables for GEL Functions

It’s possible to declare global variables outside of a function in a GEL file. A global can be declared the same way as a GEL local variable in a function. A global declared without an initial value will be initialized to 0.

A GEL file with a global will be declared when it is loaded and "undeclared" when it is unloaded. Globals must have unique names; if you declare a global with the same name as one already declared, an error will result. Also, if a target global variable or label uses the same name as a GEL global, the target value will take precedence.

See Also:

Local Variables for Functions

3.8.1.4. Function Recursion

Recursive GEL function calls are permitted.

```
RecursiveTest(num)
{
int i = num;
do
{
GEL_TextOut("@%d ", "Test Output", 0, 0, 0, i);
i++;
} while (i <= 5);
GEL_TextOut("\n", "Test Output");
if (num < 5)
RecursiveTest(num+1);
}
```

3.8.1.5. Calling GEL Functions

You can call a GEL function from anywhere you can enter a valid C expression. You can call a GEL function from any dialog box that accepts a valid C expression or from within another GEL function.

Passing arguments to a GEL function is optional. Omitted arguments assume default values.

See also:

GEL Functions: Alphabetical List

3.8.1.6. GEL return Statement

GEL supports the standard C return statement within a function. The general form is:

return [expression **];**

A function does not need to return a value; a return statement with no expression causes control, but no useful value, to be returned to the caller. The same thing happens when the end of a function is reached without encountering a return statement. The calling function can ignore a value returned by a function.

Unlike C, GEL function definitions do not specify their return type. The return type is determined during run-time.

3.8.1.7. GEL if and if-else Statements

GEL supports the standard C if and if-else statements. The general form is:

if (expression **)**
statement1**;**

if (expression **)**
statement1**;**
else
statement2**;**

In the first form, the expression is evaluated. If the expression is true (unequal to 0), statement1 is executed. In the second form, statement2 is executed if the value of the expression is 0 (zero). Each statement can be a single statement or several statements in braces.

```
if (a == 25)
b = 30;
if (b == 20)
{
a = 30;
c = 30;
}
else
{
d = 20;
}
```

3.8.1.8. GEL for Statement

GEL supports the standard C for statement. The general form is:

for (expression1**;** expression2**;** expression3 **)**
statement1**;**

The first expression is evaluated once. The value of expression1 initializes the loop counter. The second expression is evaluated before each execution of statement1. If and only if the value of expression2 is true (unequal zero), statement1 is executed. If the value of expression2 becomes false (equal to 0), the for statement is terminated. The third expression is evaluated after each execution of statement1. The value of expression3 re-initializes the loop counter. The statement can be a single statement or several statements in braces.

```
for (i = 0; i < 9; i++)
{
data1[i] = 0;
data2[i] = 0;
}
```

3.8.1.9. GEL while and do-while Statements

GEL while and do-while statements are similar to the standard C while and do-while statements, but the GEL versions do not support embedded continue statements. The general form is:

while (expression **)**
statement1**;**

do statement1
while (expression **);**

In the while statement, the expression is evaluated before each execution of statement1. If and only if the value of the expression is true (unequal zero), statement1 is executed. If the value of the expression becomes false (equal to 0), the while statement is terminated. The statement can be a single statement or several in braces.

In the do-while statement, the expression is evaluated after each execution of statement1. That is, statement1 is executed, then the expression is evaluated. If and only if the value of the expression is true (unequal zero), statement1 is executed again. If the value of the expression becomes false (equal to 0), the do-while statement is terminated. The statement can be a single statement or several in braces.

```
while (a != Count)
{
dataspace[a] = 0;
a--;
}
do
{
dataspace[a] = 0;
a--;
} while (a != Count);
```

3.8.1.10. GEL break Statement

GEL supports the standard C break statement within the iteration statements: for, while, and do-while. The general form is:

break;

A break statement terminates execution of the enclosing statement. Execution resumes at the statement following the terminated statement.

```
for (j=0; j<i; j++)
{
if (j == k)
break;
}
```

3.8.1.11. GEL Comments

GEL supports standard C and C++ style comments within a file. Comments delimited by the characters /* and */ may span several lines. Comments preceded by the characters // may continue until the end of the line. For example:

```
/* this comment
spans two lines */
// this comment continues to the end of this line
```

3.8.1.12. GEL Preprocessing Statements

GEL supports the standard #define preprocessing keyword. This is the only preprocessing keyword currently available.

A control line such as the following causes the preprocessor to replace subsequent instances of the identifier with the given sequence of tokens:

```
#define identifier token-sequence
```

Leading and trailing white spaces around the token sequence are discarded.

A control line such as the following, where there is no space between the first identifier and the open parenthesis, is a macro definition with Parameters given by the identifier list:

```
#define identifier( identifier-list ) token-sequence
```

When a macro has been defined using the #define keyword, you may use it anywhere in that file as well as in any other files that are loaded subsequently into the Code Composer Studio debugger.

3.8.1.13. GEL Expression Evaluation

GEL expressions may be used to express values in many dialog boxes, the Watch window, and the GEL toolbar, as well as within GEL functions themselves. It is in fact GEL expressions that are being evaluated wherever an expression is evaluated within Code Composer Studio IDE.

The following topics describe the enhancements that have been made to support the C++ language, and also detail the features of C++ that are not fully supported.

GEL Namespace

C++ Support

C++ Limitations

GEL Functions: Alphabetical List

Using Symbols as Expressions

3.8.2. Built-in GEL Functions

There are several built-in GEL functions that allow you to control the state of the actual/simulated target, access the actual/simulated target memory locations, and to display results in the output window.

All GEL built-in functions are preceded with the prefix GEL_ to ensure they are not confused with user-defined GEL functions. If you wish to avoid preceding all calls to GEL built-in functions with GEL_ you can define functions with your own names and call the GEL built-in functions within your functions. For example, the following GEL function allows you to call the GEL_Load() built-in functions just by typing-in Load.

```
Load(a)
{
GEL_Load(a);
}
```

Note: All built-in GEL functions and user-defined GEL functions consisting of GEL statements can be invoked directly from the console. To invoke any GEL statement or user-defined function, enter the appropriate function call in the console and press the Enter key to evaluate the expression. In this version of CCS, there are multiple console windows associated with a target; make sure the correct console window is selected as shown.

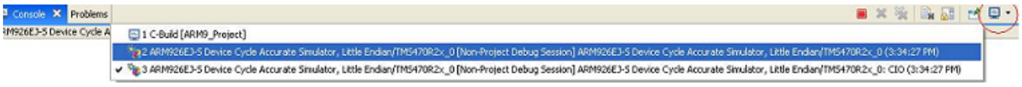

Console
Problems
RM926EJ-S Device Cycle A
1 C-Build [ARM9_Project]
2 ARM926EJ-S Device Cycle Accurate Simulator, Little Endian/TMS470R2x_0 [Non-Project Debug Session] ARM926EJ-S Device Cycle Accurate Simulator, Little Endian/TMS470R2x_0 (3:34:27 PM)
✓ 3 ARM926EJ-S Device Cycle Accurate Simulator, Little Endian/TMS470R2x_0 [Non-Project Debug Session] ARM926EJ-S Device Cycle Accurate Simulator, Little Endian/TMS470R2x_0: CIO (3:34:27 PM)

Fig. 1: Selecting the right console

Related Topics

Changes in this version

Synchronous Options

List of Built-In GEL Functions

3.8.2.1. Changes in this version

The following GEL Built-In functions are not supported.
The GEL will recognize these functions, however, no operation will be performed when they are called.

GEL_CloseWindow

GEL_Exit

GEL_OpenMemoryWindow

GEL_OpenWindow

GEL_ProjectAddSubProj

GEL_ProjectBuild

GEL_ProjectBuildConfig

GEL_ProjectClose

GEL_ProjectCreateCopyConfig

GEL_ProjectCreateDefaultConfig

GEL_ProjectLoad

GEL_ProjectRebuildAll

GEL_ProjectRebuildAllConfig

GEL_ProjectRemoveConfig

GEL_ProjectRemoveSubProj

GEL_ProjectSave

GEL_ProjectSetActive

GEL_ProjectSetActiveConfig

GEL_ProjectSetSubProjConfigSel

GEL_SrcDirAdd

GEL_SrcDirRemoveAll

GEL_StopTransferToFile

GEL_TransferToFile

GEL_TransferToFileConfg

GEL_WatchAdd

GEL_WatchDel

GEL_WatchReset

The following GEL Built-In functions are only supported in the Windows platform.

GEL_System

3.8.2.2. Synchronous Options

The following table lists the built in GEL functions and whether they are synchronous or asynchronous from GEL.

GEL Function	Synchronous from GEL	Completely Synchronous
GEL_AddInputFile	Yes	No
GEL_AddOutputFile	Yes	No
GEL_AdvancedReset	No	No
GEL_Animate	No	No

GEL_AsmStepInto	No	No
GEL_AsmStepIntoBackwards	No	No
GEL_AsmStepIntoBackwards	No	No
GEL_AsmStepOver	No	No
GEL_BreakPtAdd	Yes	No
GEL_BreakPtDel	Yes	No
GEL_BreakPtDisable	Yes	No
GEL_BreakPtReset	Yes	No
GEL_CancelTimer	Yes	Yes
GEL_ClearProfileConfiguration	Yes	Yes
GEL_DisableRealtime	No	No
GEL_DriverString	Yes	Yes
GEL_EnableClock	Yes	No
GEL_EnableRealtime	No	No
GEL_EvalOnTarget	Yes	Yes
GEL_Go	Yes (only if location given)	Yes (only if location given)
GEL_Halt	No	No
GEL_HWBreakPtAdd	Yes	No
GEL_HWBreakPtDel	Yes	No
GEL_HWBreakPtDisable	Yes	No
GEL_HWBreakPtReset	Yes	No
GEL_IgNoreUnkNownHalts	Yes	Yes
GEL_IsConnected	Yes	Yes
GEL_IsHalted	Yes	Yes
GEL_IsInRealtimeMode	Yes	Yes
GEL_Load	No	No
GEL_LoadGel	Yes	Yes
GEL_LoadProfileConfiguration	Yes	No
GEL_LowPowerRun	No	No
GEL_MapAdd	Yes	Yes
GEL_MapAddStr	Yes	Yes
GEL_MapDelete	Yes	Yes
GEL_MapOff	Yes	Yes
GEL_MapOn	Yes	Yes
GEL_MapReset	Yes	Yes
GEL_MemoryFill	Yes	Yes
GEL_MemoryLoad	Yes	Yes
GEL_MemorySave	Yes	Yes
GEL_PatchAssembly	Yes	Yes
GEL_PinConnect	Yes	Yes
GEL_PinDisconnect	Yes	Yes
GEL_PortConnect	Yes	Yes
GEL_PortDisconnect	Yes	Yes
GEL_ProbePtAdd	Yes	No
GEL_ProbePtDel	Yes	No
GEL_ProbePtDisable	Yes	No
GEL_ProbePtReset	Yes	No
GEL_RefreshWindows	No	No
GEL_RemoveDebugState	No	No
GEL_RemoveInputFile	Yes	No

GEL_RemoveOutputFile	Yes	No
GEL_Reset	Yes	Yes
GEL_Restart	Yes	Yes
GEL_RestoreDebugState	No	No
GEL_Run	No	No
GEL_RunBackwards	No	No
GEL_RunF	No	No
GEL_SetBlockResetMode	No	No
GEL_SetSimMode	Yes	No
GEL_SetStatusInterval	Yes	Yes
GEL_SetTimer	Yes	Yes
GEL_SetWaitInResetMode	Yes	Yes
GEL_SharedMemHaltOnStepOff	Yes	Yes
GEL_SharedMemHaltOnStepOn	Yes	Yes
GEL_SharedMemHaltOnWriteOff	Yes	Yes
GEL_SharedMemHaltOnWriteOn	Yes	Yes
GEL_SrcStepInto	No	No
GEL_SrcStepIntoBackwards	No	No
GEL_SrcStepOver	No	No
GEL_StepInto	No	No
GEL_StepOut	No	No
GEL_StepOver	No	No
GEL_SymbolAdd	No	No
GEL_SymbolAddRel	No	No
GEL_SymbolDisable	No	No
GEL_SymbolEnable	No	No
GEL_SymbolHideSection	No	No
GEL_SymbolLoad	No	No
GEL_SymbolLoadRel	No	No
GEL_SymbolRemove	No	No
GEL_SymbolShowSection	No	No
GEL_SyncHalt	No	No
GEL_SyncRun	No	No
GEL_SyncStepInto	No	No
GEL_SyncStepOut	No	No
GEL_SyncStepOver	No	No
GEL_System	No	No
GEL_TargetTextOut	Yes	Yes
GEL_TextOut	Yes	Yes
GEL_UnloadAllSymbols	No	No
GEL_UnloadGel	Yes	Yes
GEL_UnloadAllGels	Yes	Yes
GEL_XMDef	Yes	Yes
GEL_XMOff	Yes	Yes
GEL_XMOn	Yes	Yes

3.8.2.3. GEL Functions: Alphabetical List

See Also: The General Extension Language (GEL): An Introduction

GEL_AddInputFile()

GEL_AddOutputFile()

GEL_MapAdd()

GEL_MapAddStr()

GEL_SharedMemHaltOnWriteOn()

GEL_SharedMemHaltOnWriteOff()

GEL_AdvancedReset()

GEL_Animate()

GEL_AsmStepInto()

GEL_AsmStepOver()

GEL_BreakPtAdd()

GEL_BreakPtDel()

GEL_BreakPtDisable()

GEL_BreakPtReset()

GEL_CancelTimer()

GEL_ClearProfileConfiguration()

GEL_Connect()

GEL_DisableFileOutput()

GEL_DisableRealtime()

GEL_Disconnect()

GEL_EnableClock()

GEL_EnableFileOutput()

GEL_EnableRealtime()

GEL_EnableZeroFill()

GEL_EvalOnTarget()

GEL_Go()

GEL Grammar

GEL_Halt()

GEL_HandleTargetError()

GEL_HWBreakPtAdd()

GEL_HWBreakPtDel()

GEL_HWBreakPtDisable

GEL_HWBreakPtReset()

GEL_IsConnected()

GEL_IsHalted()

GEL_IsInRealtimeMode()

GEL_Load()

GEL_LoadGel()

Gel_LoadProfileConfiguration()

GEL_LowPowerRun()

GEL_MapDelete()

GEL_MapOff()

GEL_MapOn()

GEL_MapReset()

GEL_MemoryFill()

GEL_MemoryLoad()

GEL_MemoryLoadData()

GEL_MemorySave()

GEL_MemorySaveData()

GEL_MemoryListSupportedTypes()

GEL_PatchAssembly()

GEL_PinConnect()

GEL_PortConnect()

GEL_PortDisconnect()

GEL_ProbePtAdd()

GEL_ProbePtDel()

GEL_ProbePtDisable

GEL_ProbePtReset()

GEL_RefreshWindows()

GEL_RemoveDebugState()

GEL_RemoveInputFile()

GEL_RemoveOutputFile()

GEL_Reset()

GEL_Restart()

GEL_RestoreDebugState()

GEL_Run()

GEL_RunF()

GEL_SetBlockResetMode()

GEL_SetTimer()

GEL_SetSimMode()

GEL_SetWaitInResetMode()

GEL_SetSemihostingMainArgs()

GEL_SharedMemHaltOnStepOff()

GEL_SharedMemHaltOnStepOn()

GEL_StepInto()

GEL_StepOut()

GEL_StepOver()

GEL_SrcStepInto()

GEL_SrcStepOver()

GEL_SyncHalt()

GEL_SyncRun()

GEL_SyncStepInto()

GEL_SyncStepOut()

GEL_SyncStepOver()

GEL_SymbolAdd()

GEL_SymbolAddRel()

GEL_SymbolHideSection()

GEL_SymbolRemove()

GEL_SymbolShowSection()

GEL_SymbolLoad()

GEL_SymbolLoadRel()

GEL_System()

GEL_TargetTextOut()

GEL_TextOut()

GEL_Trace()

GEL_VerifyProgram()

GEL_UnloadAllSymbols()

GEL_UnloadGel()

GEL_UnloadAllGels()

OnChildRunning()

OnFileLoaded()

OnHalt

OnPreFileLoaded()

OnReset()

OnResetDetected()

OnRestart()

OnTargetConnect()

Startup()

3.8.2.3.1. GEL_AddInputFile()

To create a breakpoint at a specified address for use with File I/O input, use this syntax:

Syntax

GEL_AddInputFile(programAddr**, "**connectFileName**",** Format**, "**dataAddr**" [, "**length**"] [,** page**] [,** wraparound**] [, "**condition**"]);**

To create a breakpoint at a specified line in a specified file for use with File I/O input, use this syntax:

Syntax

GEL_AddInputFile("SrcFileName**",** lineNumber**, "**connectFileName**",** Format**, "**dataAddr**" [, "**length**"] [,** page**] [,** wraparound**] [, "**condition**"]);**

Parameters:

programAddr is a value specifying an expression that evaluates to the location of the breakpoint. This expression should **not** be enclosed in quotation marks. If programAddr is not specified, the current address is used.

SrcFileName is a string value specifying the filename of the source file to place the breakpoint in. The file path need not be provided. The file must have been compiled as part of the current set of loaded symbols in order to be properly resolved.

LineNumber is a long integer value specifying the line in srcFileName at which to place the breakpoint.

ConnectFileName is a string value specifying the path and filename of the data input file to connect the breakpoint to. A double backslash escape sequence is required to ensure that a backslash is inserted into the filename; for example: "c:\\ti\\myprojects\\sine.dat".

Format is an integer value representing the file format. The following choices are available:

1	*.dat	Hex
2	*.dat	Integer
3	*.dat	Long
4	*.dat	Float
5	*.out	COFF

dataAddr is a string value specifying an expression that evaluates to an address; this is the address that data from the file will be written to.

length(optional) is a string value that specifies an expression evaluating to the number of words to be written to the target, starting at dataAddr, each time the breakpoint is hit. The default value is 1.

pagean integer value that specifies the page on which dateAddr is located, where the File I/O transfers will read/write to. On targets that do not support paged memory, specify 0. Otherwise, the following choices are available: 0 (Program memory), 1 (Data memory) or 2 (I/O space)

wraparound(optional) is a numeric value specifying whether or not reading should continue from the beginning of the input file once the end is hit. The default value is 0.

condition(optional) is a string value specifying the condition under which the breakpoint should be executed (that is, the FileI/O transfer should take place). The default value is "".

Description

This function has two uses. The first format listed creates a breakpoint at the specified address, opens the specified input file, connects it to the breakpoint, and places the resulting input FileI/O item in the Playing state.

The second format listed creates a breakpoint at the specified line in the specified source file, opens the specified input file, connects it to the breakpoint, and places the resulting input FileI/O item in the Playing state.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: No

Related Topics

GEL_AddOutputFile

GEL_RemoveInputFile

3.8.2.3.2. GEL_AddOutputFile()

Syntax

To create a breakpoint at a specifed address for use with FileI/O output:

GEL_AddOutputFile(programAddr**, "**connectFileName**",** Format**, "**dataAddr**" [, "**length**"] [,** page**] [, "**condition**"])**

To create a breakpoint at the specifed line in the specified source file for use with FileI/O output:

GEL_AddOutputFile("srcFileName**",** lineNumber**, "**connectFileName**",** Format**, "**dataAddr**" [, "**length**"] [,** page**] [, "**condition**"])**

Parameters

SrcFileName is a string value specifying the path and filename of the source file to place the breakpoint in. The file path need not be provided. The file must have been compiled as part of the current set of loaded symbols in order to be properly resolved.

lineNumber is a long integer value specifying the line in srcFileName at which to place the breakpoint.

programAddr is a value specifying an expression that evaluates to the location of the breakpoint. This expression must **not** be enclosed in quotation marks. If programAddr is not specified, the current address is used.

ConnectFileName is a string value specifying the path and filename of the data output file to which to connect the breakpoint. A double backslash escape sequence is required to ensure that a backslash is inserted into the filename. For example: "c:\\ti\\myprojects\\out.dat".

format is an integer value representing the file format. The following choices are available:

1	*.dat	Hex
2	*.dat	Integer
3	*.dat	Long
4	*.dat	Float
5	*.out	COFF

dataAddr is a string value specifying an expression that evaluates to an address; this is the address from which data will be read from to write to the output file.

length(optional) is a string value that specifies an expression evaluating to the number of words to be read from the target, starting at dataAddr, each time the breakpoint is hit. The default value is 1.

pagean integer value that specifies the page on which dateAddr is located, where the File I/O transfers will read/write to. On targets that do not support paged memory, specify 0. Otherwise, the following choices are available: 0 (Program memory), 1 (Data memory) or 2 (I/O space)

condition(optional) is a string value specifying the condition under which the breakpoint should be executed (that is, the FileI/O transfer should take place). The default value is "".

Description

This function has two uses. The first format listed creates a breakpoint at the specified address, opens the specified output file, connects it to the breakpoint, and places the resulting output FileI/O item in the Playing state.

The second format listed creates a breakpoint at the specified line in the specified source file, opens the specified output file, connects it to the breakpoint, and places the resulting output FileI/O item in the Playing state.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: No

Related Topics

GEL_AddInputFile

GEL_RemoveOutputFile

3.8.2.3.3. GEL_AdvancedReset()

Resets processor.

Syntax

GEL_AdvancedReset("string"**);**

Parameters

System Reset: Resets the device

Description

This GEL function issues any possible reset other than the default reset for the connected hardware device.

Synchronous

Synchronous from GEL: No

Completely Synchronous: No

Related Topics

GEL_Reset

GEL_Restart

GEL_Run

GEL_Halt

OnReset

3.8.2.3.4. GEL_Animate()

Animates the target application.

Syntax

GEL_Animate();

Parameters

None.

Description

This function starts animating the target application.

Synchronous

Synchronous from GEL: No

Completely synchronous: No

Related Topics

GEL_Go

GEL_Halt

GEL_Run

3.8.2.3.5. GEL_AsmStepInto()

Executes the next assembly instruction.

Syntax

GEL_AsmStepInto([count] **);**

Parameters

count(optional) specifies the number of times the command is to be repeated. If no count is specified, GEL_ASMStepInto executes a single assembly instruction.

Description

Executes the next assembly instruction, whether source is available or not. Repeats count times. Optionally, you can enable/disable automatic focus change to a Disassembly window, in assembly or mixed mode, to show the current assembly instruction.

Synchronous

Synchronous from GEL: No

Completely synchronous: No

Example

GEL_AsmStepInto();

Related Topics

GEL_SrcStepInto

GEL_SrcStepOver

GEL_AsmStepOver

3.8.2.3.6. GEL_AsmStepOver()

Steps over the next assembly instruction.

Syntax
GEL_AsmStepOver([count]?**);**

Parameters

count:(optional) specifies the number of times the command is to be repeated. If no count is specified, steps over a single assembly instruction.

Description

In the Assembly or mixed mode, steps over a single assembly instruction. Repeats count times. If the instruction is an assembly subroutine, executes the assembly subroutine and then halts after the assembly function returns. In source mode, may not appear to move the source cursor at all.

Synchronous

Synchronous from GEL: No

Completely synchronous: No

Related Topics

GEL_SrcStepInto

GEL_SrcStepOver

GEL_AsmStepInto

3.8.2.3.7. GEL_BreakPtAdd()

Adds a breakpoint.

Syntax

GEL_BreakPtAdd(address [**,"**Condition**"**] **);**

Parameters

address identifies the location of the breakpoint.

Condition(optional) specifies the condition used in the conditional breakpoint If you specify a condition, it must be enclosed in quotation marks.

Description

This function sets a software breakpoint at a specific address. If a condition is specified, the breakpoint becomes a conditional breakpoint and execution stops only if the condition evaluates to true. The address can be any GEL expression that evaluates to a program address, or a string that specifies a symbolic location.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: No

Example

```
GEL_BreakPtAdd(0x2000);
GEL_BreakPtAdd(TargetLabel + 100);

GEL_BreakPtAdd("main");
GEL_BreakPtAdd(0x2000, "a < b");
```

Related Topics

GEL_BreakPtDel

GEL_BreakPtReset

GEL_BreakPtDisable

3.8.2.3.8. GEL_BreakPtDel()

Delete a breakpoint.

Syntax

GEL_BreakPtDel(address **);**

Parameters

address identifies the location of the breakpoint.

Description

This function clears a software breakpoint at a specific address. If there is no software breakpoint set at address, nothing happens. The address can be an absolute address, any C expression, the name of a C function, or the name of an assembly language label.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: No

Example

GEL_BreakPtDel(0x2000);
GEL_BreakPtDel(TargetLabel + 100);

Related Topics

GEL_BreakPtAdd

GEL_BreakPtReset

3.8.2.3.9. GEL_BreakPtDisable

Disables a software breakpoint at the specified address.

Format

GEL_BreakPtDisable(address**);**

Parameters

address specifies the location of the software breakpoint.

Description

The software breakpoint at the specified address is disabled. If the breakpoint does not exist or is already disabled, no action is taken. This function will not disable the breakpoint if it is a hardware breakpoint.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: No

Example

GEL_BreakPtDisable(0x2000);

GEL_BreakPtDisable(main + 100);

Related Topics

GEL_BreakPtAdd

GEL_BreakPtDel

GEL_BreakPtDisable

GEL_ClearProfileConfiguration

3.8.2.3.10. GEL_BreakPtReset()

Clears all breakpoints.

Syntax

GEL_BreakPtReset();

Parameters

None.

Description

This function clears all software breakpoints.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: No

Example

GEL_BreakPtReset();

Related Topics

GEL_BreakPtAdd

GEL_BreakPtDel

3.8.2.3.11. GEL_CancelTimer

Cancels a previously set GEL timer.

Format

GEL_CancelTimer(timer_id);

Parameters

timer_id specifies the same timer_id that was used to set the timer.

Description

Cancels a timer that was previously set with GEL_SetTimer. The timer_id specified in the second parameter of GEL_SetTimer uniquely identifies the timer, and that same value used in this function cancels that specific timer.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: Yes

Example

GEL_CancelTimer(1);

Related Topics

GEL_SetTimer

3.8.2.3.12. GEL_ClearProfileConfiguration

Clears the profile configuration.

Format

GEL_ClearProfileConfiguration( );

Parameters

None

Description

This function clears all profile configuration information, including activities, ranges, control points, and custom data collection options.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: Yes

Example

GEL_ClearProfileConfiguration( );

Related Topics

Gel_LoadProfileConfiguration

3.8.2.3.13. GEL_DisableFileOutput()

Disables writing GEL Text output to a file

Syntax

GEL_DisableFileOutput();

Parameters

none

Description

Disables writting GEL Text output to file

Synchronous

Synchronous from GEL: Yes

Completely synchronous: Yes

Example

GEL_DisableFileOutput();

Related Topics

GEL_EnableFileOutput

GEL_TextOut

GEL_Trace

3.8.2.3.14. GEL_DisableRealtime()

Disables real-time debugging.

Syntax

GEL_DisableRealtime();

Parameters

None.

Description

This function disables real-time debugging.

Synchronous

Synchronous from GEL: No

Completely synchronous: No

Example

GEL_DisableRealtime();

Related Topics

GEL_EnableRealtime

3.8.2.3.15. GEL_EnableClock()

Enables or disables the profile clock.

Syntax

GEL_EnableClock([bool enable]**);**

Parameters

[optional]

0 disables the clock.

1 enables the clock

blank enables the clock

Description

This function is used to enable or disable the profile clock.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: Yes

Example

GEL_EnableClock(); //Enable Clock

GEL_EnableClock(0); //Disable Clock

GEL_EnableClock(1); //Enable Clock

3.8.2.3.16. GEL_EnableFileOutput()

Enables writing GEL Text output to a file

Syntax

GEL_EnableFileOutput("filePath" [, appendToEnd] [, teeToConsole] **);**

Parameters

filePath is the path of the file to which output should be written.

appendToEnd (optional) indicates whether to append to or overwrite a file. 0 - file content should be overwritten. 1 - append content. The default behaviour is to append.

teeToConsole (optional) indicates whether to send the output to the console as well. 0 - do not send the output. 1 - do send the output. The default behaviour is to send the output to the console.

Description

Enables writting GEL Text output generated by calls such as GEL_TextOut to a file specified by the filePath without affecting any other behaviour

Synchronous

Synchronous from GEL: Yes

Completely synchronous: Yes

Example

```
GEL_EnableFileOutput("c:\\gel_out\\gel_log.txt");
GEL_EnableFileOutput("c:\\gel_out\\gel_log.txt" , 1);
GEL_EnableFileOutput("c:\\gel_out\\gel_log.txt", 0, 1);
```

Related Topics

GEL_TextOut

GEL_Trace

GEL_DisableFileOutput

3.8.2.3.17. GEL_EnableRealtime()

Enables real-time debugging.

Syntax

GEL_EnableRealtime();

Parameters

None.

Description

This function enables real-time debugging.

Synchronous

Synchronous from GEL: No

Completely synchronous: No

Example

GEL_EnableRealtime();

Related Topics

GEL_DisableRealtime

3.8.2.3.18. GEL_EvalOnTarget()

Evaluate a GEL expression on a target

Syntax

GEL_EvalOnTarget(target_path_name, expression, synchronous = true **);**

Parameters

target_path_name is the path name of the target on which the expression will be evaluated

expression is the expression to be evaluated

synchronous, if true, GEL_EvalOnTarget won't return until expression has completed. If false, it will start the evaluation and return immediatel

Description

The GEL_EvalOnTarget function evaluates the expression on the target whose path name is target_path_name. If synchronous, it returns the result of the underlying expression. Otherwise zero is returned

3.8.2.3.19. GEL_Go()

Runs to specified address.

Syntax

GEL_Go([address] **);**

Parameters

address(optional) is the stop address.

Description

The GEL_Go function executes code up to a specific point in the program. The address Parameters is treated as a program-memory address. If you do not supply an address, then GEL_Go becomes equivalent to the GEL_Run GEL function.

Synchronous

Synchronous from GEL: Yes, only if location is given

Completely synchronous: Yes, only if location is given

Example

GEL_Go();
GEL_Go(main);

Related Topics

GEL_Run

GEL_Halt

3.8.2.3.20. GEL_Halt()

Stops execution.

Syntax

GEL_Halt();

Parameters

None.

Description

This function halts the target program if it is executing.

Synchronous

Synchronous from GEL: No

Completely synchronous: No

Example

GEL_Halt();

Related Topics

GEL_Go

GEL_IsHalted

GEL_Run

GEL_RunF

3.8.2.3.21. GEL_HWBreakPtAdd()

Adds a hardware breakpoint.

Syntax

GEL_HWBreakPtAdd(address [**,**count]**);**

Parameters

address is the location of the breakpoint. If the address is a string Parameters, a symbolic breakpoint will be set at that location and re-evaluated whenever symbols change. Otherwise, a breakpoint will be set at the specified address.

count(optional) is the number of times to execute before breakpoint is triggered. The default value is 1.

Description

This function adds a new, enabled type 1 hardware breakpoint at the specified address, if the target supports it.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: No

Example

```
GEL_HWBreakPtAdd( 0x2000 );
GEL_HWBreakPtAdd( PC + 100 );
GEL_HWBreakPtAdd( "main" );
GEL_HWBreakPtAdd( 0x2000, 5 );
```

Related Topics

GEL_HWBreakPtDel

GEL_HWBreakPtReset

GEL_HWBreakPtDisable

3.8.2.3.22. GEL_HWBreakPtDel()

Removes a hardware breakpoint.

Syntax

GEL_HWBreakPtDel(address**);**

Parameters

address is the memory location of the hardware breakpoint to be removed.

Description

This function removes any hardware breakpoint currently set at the specified address, if the target supports it.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: No

Related Topics

GEL_HWBreakPtAdd

GEL_HWBreakPtReset

3.8.2.3.23. GEL_HWBreakPtDisable

Disables a hardware breakpoint at the specified address.

Format

GEL_HWBreakPtDisable(address**);**

Parameters

address specifies the location of the hardware breakpoint.

Description

The hardware breakpoint at the specified address is disabled. If the breakpoint does not exist or is already disabled, no action is taken. This function does not disable software breakpoints.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: No

Example

GEL_HWBreakPtDisable(0x2000);

GEL_HWBreakPtDisable(main + 100);

Related Topics

GEL_HWBreakPtAdd()

GEL_HWBreakPtDel()

GEL_HWBreakPtReset()

GEL_BreakPtDisable

3.8.2.3.24. GEL_HWBreakPtReset()

Removes all hardware breakpoints.

Syntax

GEL_HWBreakPtReset();

Parameters

None.

Description

This function removes all hardware breakpoints currently set, whether they are enabled or disabled.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: No

Related Topics

GEL_HWBreakPtAdd

GEL_HWBreakPtDel

3.8.2.3.25. GEL_IsConnected

GEL built-in function that can be called at any time to determine if the target is connected.

Syntax

GEL_IsConnected()

Parameters

None

Description

The function returns 1 if the target is connected and returns 0 otherwise.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: Yes

Related Topics

GEL_Connect

GEL_Disconnect

3.8.2.3.26. GEL_IsHalted

GEL built-in function that can be called at any time to determine if the target is halted.

Syntax

GEL_IsHalted()

Parameters

None

Description

The function returns 1 if the target is halted and returns 0 otherwise.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: Yes

Related Topics

3.8.2.3.27. GEL_IsInRealtimeMode

Called every time a target is connected. GEL built-in function that can be called at any time to determine if the target is in realtime mode.

Syntax

GEL_IsInRealtimeMode()

Parameters

None

Description

The function returns 0 is the DSP is currently in Stop mode and 1 if it is in Real Time Mode. Some targets cannot enter Real Time mode; for those cases, the function will always return 0. (See real-time mode)

Synchronous

Synchronous from GEL: Yes

Completely synchronous: Yes

Example

```
OnTargetConnect()

{ if( GEL_IsInRealtimeMode() )

{

// The target was connected in realtime mode

}

else

{

//The target was connected in stop mode

}
```

Related Topics

3.8.2.3.28. GEL_Load()

Loads a program

Syntax

GEL_Load("fileName" [**,"**cpuName**"**] [**,"**boardName**"**] **);**

Parameters

filenamenames is the file name of program file to be loaded. The fileName must be enclosed in quotation marks.

cpuName(optional) names the CPU on which to load the file (useful in a multiprocessor environment). The cpuName must be enclosed in quotation marks.

boardName(optional) is the text string specified as the Board Name in the target configuration. The boardName must be enclosed in quotation marks.

Description

This function downloads both program code and data onto the target at the addresses specified in the file. Symbols are loaded into a symbol table maintained by the debugger on the host. The symbols are loaded at the code and data addresses specified in the file.

Supported file formats

- **COFF**
- **ELF**
- **S-Record**
- **Intel Hex**
- **Tektronix Hex**

If the file is not in the current directory, provide a full path name within the string. A double backslash escape sequence is required to ensure that you get a backslash into the fileName.

The cpuName must match the CPU name as configured in the multiprocessor setup. In a single processor system, you do not need to fill this field.

Synchronous

Synchronous from GEL: No

Completely synchronous: No

Example

```
GEL_Load("c:\\mydir\\myfile.out", "cpu_a", "Emulator");
```

Related Topics

GEL_SymbolAdd

GEL_SymbolAddRel

GEL_SymbolLoad

GEL_SymbolLoadRel

GEL_SymbolRemove

GEL_UnloadAllSymbols

GEL_VerifyProgram

3.8.2.3.29. Gel_LoadProfileConfiguration

Loads a profile configuration from a file.

Format

GEL_LoadProfileConfiguration(sFileName**)**;

Parameters

sFileNamenames the file to load the configuration from.

Description

This function loads a saved profile configuration, including activities, ranges, control points, and custom data collection options.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: No

Example

GEL_LoadProfileConfiguration("C:\\config.ini");

Related Topics

GEL_ClearProfileConfiguration

3.8.2.3.30. GEL_LoadGEL()

Loads a GEL file.

Syntax

GEL_LoadGel("fileName**");**

Parameters

fileName names the GEL file to be loaded. The fileName must be enclosed in quotation marks.

Description

This function loads the specified GEL file.

If the file is not in the current directory, provide a full path name within the string. A double backslash escape sequence is required to ensure that you get a backslash into the fileName Parameters.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: Yes

Example

GEL_LoadGel("c:\\mydir\\myfile.gel");

Related Topics

GEL_UnloadGel

GEL_UnloadAllGels

3.8.2.3.31. GEL_LowPowerRun()

Runs target in a low power mode.

Syntax

GEL_LowPowerRun();

Parameters

None

Description

This GEL function allows the target to run in low power mode.

Synchronous

Synchronous from GEL: No

Completely Synchronous: No

Related Topics

GEL_Run

3.8.2.3.32. GEL_MapAdd()

Adds to the memory map.

Syntax

GEL_MapAdd(address**,** page**,** length**,** readable**,** writeable**);**

Parameters

addressis the starting address of a range in memory. This Parameters can be an absolute address, any C expression, the name of a C function, or an assembly language label.

pageidentifies the type of memory to fill: 0 (Program memory), 1 (Data memory) or 2 (I/0 space).

For processors which do not have more than one type of memory, use 0 for this Parameters. For simulated targets, I/O Space Parameters is not supported.

lengthdefines the length of the range. This Parameters can be any C expression.

readabledefines whether the memory range is readable: 0 (Not readable) or 1 (Readable)

writeabledefines whether the memory range is writeable: 0 (Not writeable) or 1 (Writeable)

Description

This function adds read/write permission for a range of target memory to the memory map. If the range overlaps an existing entry, the attributes of the new range take precedence in the memory map.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: Yes

Example

GEL_MapAdd(0x1000, 0, 0x300, 1, 1);

Related Topics

GEL_MapAddStr

GEL_MapDelete

GEL_MapOn

GEL_MapOff

GEL_MapReset

3.8.2.3.33. GEL_MapAddStr()

Adds to the memory map.

Syntax

GEL_MapAddStr(address**,** page**,** length**, "**attribute**",** waitstate **);**

Parameters

address	Starting address of a range in memory. This Parameters can be an absolute address, any C expression, the name of a C function, or an assembly language label.
page	Identifies the type of memory to fill: 0 (Program memory), 1 (Data memory) or 2 (I/0 space) For processors which do not have more than one type of memory, use 0 for this Parameters. For simulated targets, the I/O Space option is not supported.
length	Defines the length of the range. This Parameters can be any C expression
attribute	Defines one or more attributes for the specified memory range. The attributes must be enclosed in quotation marks. See Predefined Attribute Strings. Additional attributes may be supported for your device driver. More than one attribute can be specified. Use the vertical bar "|" character to separate multiple attributes. For example, "R|W|P" is equivalent to "IOPORT". Access size can be specified using "ASn", where n is the access size in bytes. For example, "R|AS4" specifies 32 bits ROM. The attribute "SHnC" can be used to define a block of shared memory.
waitstate	Defines the number of waitstates. When reading from or writing to slower external memory, the CPU waits one extra clock cycle for every waitstate. The waitstate Parameters accepts any non-negative integer value (for example: 0, 1, 2, 3).

Description

This function adds one or more attributes for a range of target memory to the memory map. If the range overlaps an existing entry, the attributes of the new range take precedence in the memory map.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: Yes

Predefined Attribute Strings

String	Description	String	Description
R	Read	NONE	No memory/protected
W	Write	RAM	Read and write †
P	Port	ROM	Read only †
EX	External	WOM	Write only
EM	Emulator	INPORT	Port read only
PR	Programmable	OUTPORT	Port write only
ER	Erasable	IOPORT	Port read and write
DA	Dual access	SARAM	Single access RAM
ASn	Access size	DARAM	Dual access RAM
SHnC	Shared †	FLASH	Flash ROM
CACHE	Cache †	EXRAM	External RAM
TX	Text	EXROM	External ROM
MN	Monitor	EPROM	Erasable write-able EPROM
SA	Single access	MONITOR	Monitor ROM
FL	Flash	PRAM	Program RAM
MR	Memory mapped	PROM	Program ROM
NULL	NULL	NULL	NULL

†Defining Shared Memory describes how to use these attributes to add a range of shared memory to the memory map.

Example

```
GEL_MapAddStr(0x1000, 0, 0x300, "RAM", 0);
GEL_MapAddStr(0x1000, 0, 0x300, "R|W|P", 0);
GEL_MapAddStr(0x1000, 0, 0x300, "R|W|AS1", 0);// 8 bits RAM
GEL_MapAddStr(0x1000, 0, 0x300, "R|AS4", 0); // 32 bits ROM
```

Related Topics

GEL_MapAdd

GEL_MapDelete

GEL_MapOn

GEL_MapOff

GEL_MapReset

3.8.2.3.34. GEL_MapDelete()

Deletes from the memory map.

Syntax

GEL_MapDelete(address**,** page**);**

Parameters

addressidentifies the memory range already defined in the memory map that is to be deleted from the memory map. Address can be any valid address in the memory map range that is to be deleted. This Parameters can be an absolute address, any C expression, the name of a C function, or an assembly language label.

pageidentifies the page that the memory range is on: 0 (Program memory), 1 (Data memory) or 2 (I/O space)

For processors that do not have more than one type of memory, use 0 for this Parameters. For simulated targets, the I/O space Parameters is not supported.

Description

This function deletes a range of memory from the memory map. When deleted, the Code Composer Studio debugger does not read or write from/to the target. If you display a memory location that is not readable, the debugger does not display the value on the target, instead it displays the default value.

GEL_MapDelete deletes an entire memory map range. Although you specify a specific memory address as a parameter, in fact it deletes the entire range in which this memory address is located. For example, if you have a range of (0x000 – 0x1007), specifying any address in that range within the GEL_MapDelete causes the whole range to be deleted.

If you wanted to delete a whole range of memory map entries, you could re-add them with another GEL_MapAdd statement, and then delete the newly added range. Suppose you had a map that looked like:

0x1000 – 0x1007

0x1009 – 0x100e

0x1010 – 0x1017

0x1020 – 0x102e

You could delete everything between 0x1000 and 0x102f by first adding a new range between 0x1000 and 0x102f and then call GEL_MapDelete with any address falling in that range.

GEL_MapAdd(0x1000, page, 0x2f, readable, writable)

/*causes existing map entries to be combined */

GEL _MapDelete(0x1000, page) /* deletes this new entry */

Synchronous

Synchronous from GEL: Yes

Completely synchronous: Yes

Example

GEL_MapDelete(0x1000, 0);

Related Topics

GEL_MapAdd

GEL_MapOn

GEL_MapOff

GEL_MapReset

3.8.2.3.35. GEL_MapOff()

Disables a memory map.

Syntax

GEL_MapOff();

Parameters

None.

Description

This function disables memory mapping. Note that disabling memory mapping can cause bus fault problems in the target because the Code Composer Studio debugger may attempt to access nonexistent memory. On power up, the memory map is turned off by default.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: Yes

Example

GEL_MapOff();

Related Topics

GEL_MapAdd

GEL_MapOn

GEL_MapDelete

GEL_MapReset

3.8.2.3.36. GEL_MapOn()

Enables memory mapping.

Syntax

GEL_MapOn()

Parameters

None.

Description

This function enables memory mapping. The Code Composer Studio debugger does not attempt to read from a map segment that is not readable or attempt to write to a map segment that is not writeable. When mapping is first turned on, the entire memory range is assumed to have no reading or writing capabilities. You must add memory sections (using the GEL_MapAdd() function) to allow the debugger to access valid sections. On power up, memory mapping is turned off by default.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: Yes

Example

GEL_MapOn();

Related Topics

GEL_MapAdd

GEL_MapOff

GEL_MapDelete

GEL_MapReset

3.8.2.3.37. GEL_MapReset()

Resets the memory map.

Syntax

GEL_MapReset()

Parameters

None.

Description

This function resets the memory map by making all memory non-readable and non-writeable.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: Yes

Example

GEL_MapReset();

Related Topics

GEL_MapAdd

GEL_MapOff

GEL_MapOn

GEL_MapDelete

3.8.2.3.38. GEL_MemoryFill()

Fills a block of memory.

Syntax

GEL_MemoryFill(startAddress, page, length, pattern [, patternSizeType])

Parameters

startAddressis the first address in the block.

page identifies the type of memory to fill: 0 (Program memory), 1 (Data memory) or 2 (I/0 space).

For processors that do not have more than one type of memory, use 0 for this Parameters. For simulated targets, the I/O space Parameters is not supported.

length defines the number of words to fill.

pattern is the value that is placed in each word in the block.

patternSizeType (optional) specifies the size of pattern that CCStudio should interpret. If not provided, pattern size will be interpreted as being the word size on the memory page on which the GEL_MemoryFill operation is taking place. Acceptable values for patternSizeType are as follows:

Target specific types: size of the type depends on the target used

0x00 - Char

0x01 - Unsigned Char

0x02 - Short

0x03 - Unsigned Short

0x04 - Enum

0x06 - Int

0x07 - Unsigned Int

0x08 - Long

0x09 - Unsigned Long

0x0B - Pointer

Target independent signed types

0x0C - 8 bit integer

0x0E - 16 bit integer

0x10 - 24 bit integer

0x12 - 32 bit integer

0x14 - 40 bit integer

0x16 - 48 bit integer

0x18 - 64 bit integer

Target independent unsigned types

0x0D - 8 bit unsigned integer

0x0F - 16 bit unsigned integer

0x11 - 24 bit unsigned integer

0x13 - 32 bit unsigned integer

0x15 - 40 bit unsigned integer

0x17 - 48 bit unsigned integer

0x19 - 64 bit unsigned integer

Large and floating point types

0x1E - long long

0x1F - unsigned long long

0x20 - float

0x22 - 32 bit IEEE single precision float

0x24 - double

0x26 - 64 bit IEEE double precision float

0x28 - long double

Description

This function can be used to fill a block of target memory with a specified pattern.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: Yes

Example

// Fill memory 0x1000 on page 0 with 0x1000 instances of 0xa5a5 where 0xa5a5 is interpreted by CCS as a 32-bit unsigned integer.

GEL_MemoryFill(0x1000, 0, 0x100, 0xa5a5, 0x13);

Related Topics

GEL_MemoryLoad

GEL_MemorySave

3.8.2.3.39. GEL_MemoryLoad()

Loads a block of memory from a file.

Syntax

GEL_ MemoryLoad(startAddress**,** page**,** length**, "**fileName**", "**[bitsize]**", "**[swap]**")**

Parameters

startAddress is the first address in the block.

page identifies the type of memory to fill: 0 (Program memory), 1 (Data memory) or 2 (I/O space)

For processors that do not have more than one type of memory, use 0 for this Parameters. For simulated targets, the I/O space Parameters is not supported.

length defines the number of words to fill.
Note: This parameter is ignored when loading binary (.bin) and Coff files. The entire file is loaded regardless.

fileName names the file to store the target data. The fileName must be enclosed in quotation marks.

bitsize is optional if the file is not raw binary. If the file is raw binary, it indicates the bit size of the data type to interpret the data as before loading to target. This is important when doing automated endianess conversion when loading data to the target. Since, the file data is assumed to be little endian; when loading on to a big endian target some bytes may need to be swapped.

8 - Read data one byte at time. No need to swap

16 - Read data two bytes at a time. Swap bytes when loading to big endian target

32 - Read data 4 bytes at a time. Swap bytes when loading to big endian target

swap - swap the data before applying automated endiannes conversion to load to target. This effectivly disables the automated endiannes conversion.

Description

You can use GEL_MemoryLoad() to load a block of target memory from a specified file. The block of data is specified by the startAddress, page, and length. If the filename contains a *.out for the file extension, COFF format is used; otherwise, if we are loading from data file (.dat) Code Composer Studio debugger uses the header information in the file to determine the file format. If we are reading from a binary file (.bin) the assumption is that the data is stored in little endian format to perform automated endiannes conversion when loading to target. The bitsize and swap options could be used to further tweak how the data is interpreted and load from a big endian format. See GEL_MemoryLoad2 for additional formats suppport

Synchronous

Synchronous from GEL: Yes

Completely synchronous: Yes

Example

GEL_MemoryLoad(0x1000, 1, 0x100, "c:\\mydir\\myfile.dat");

Related Topics

GEL_MemorySave

GEL_MemoryFill

GEL_MemorySave2

GEL_MemoryLoad2

3.8.2.3.40. GEL_MemorySave()

Saves a block of memory to file.

Syntax

GEL_MemorySave(startAddress**,** page**,** length**, "**fileName**"**[**,**io_Format][**,**append]**[,bitsize][,swap])**

Parameters

startAddress is the first address in the block.

page identifies the type of memory to fill: 0 (Program memory), 1 (Data memory) or 2 (I/O space)

For processors that do not have more than one type of memory, use 0 for this Parameters. For simulated targets, I/O Space Parameters is not supported. GEL does not give warning messages for invalid parameters such as invalid page values.

length defines the number of words to fill.

fileName names the file to store the target data. The fileName must be enclosed in quotation marks.

io_format is an integer that represents the format in which memory words will be written to the specified output file. The default output is 1 (hexadecimal). It accepts the following Parameters:

1	*.dat	Hex
2	*.dat	Integer
3	*.dat	Long
4	*.dat	Float
5	*.out	COFF
6	*	Addressable unit
7	*	Use Header
8	*.bin	Raw Binary

append represents whether memory to be saved should overwrite the contents of the specified file (0) or append to the end of the file (any nonzero value).

0	Overwrite the contents of the specified file
Any nonzero value	Append to the end of the file

Appending will neither modify existing nor create file header information. The append feature is **not** supported for COFF formatted files.

bitsize is option if the file is not raw binary. If we are save to a raw binary file (.bin). it indicates the bit size of the data type to interpret the data as when saving to file. Since, data is saved in a little endian format in the file; this is important when doing automated endiannes conversion while saving the data.

8 - Save data one byte at time. No need to swap

16 - Save data two bytes at a time. Swap bytes when reading from big endian target to convert to little endian

32 - Read data 4 bytes at a time. Swap bytes when reading from big endian target to convert to little endian

64 - Read data 8 bytes at a time. Swap bytes when reading from big endian target to convert to little endian

swap - swap the data before applying the automated endianness conversion. This effectivly disables the automated endiannes conversion. Set swap to true to save data as big endian on the host side

Description

This function can be used to save a block of target memory to a specified file. The block of data is specified by the startAddress, page, and length. If the filename contains a *.out for the file extension, COFF format is used; otherwise, When data is saved in a raw binary (.bin) it is written out in little endian format independant of the target endianness. The bitsize and swap options could be used to further tweak how the binary data is saved and save to big endian format instead. For support for additional formats see GEL_MemorySave2

Synchronous

Synchronous from GEL: Yes

Completely synchronous: Yes

Examples

GEL_MemorySave(0x1000, 1, 0x100, "c:\\mydir\\myfile.dat");

This example appends the block of memory to saved_memory.txt and formats the memory as integers:

GEL_MemorySave(0x100, 0, 0x10, "saved_memory.txt",2, 1);

Related Topics

GEL_MemoryLoad

GEL_MemoryFill

GEL_MemoryLoad2

GEL_MemorySave2

3.8.2.3.41. GEL_PatchAssembly()

Patches the memory with assembly patch string.

Syntax

GEL_PatchAssembly(address**,** page**, "**patchString**");**

Parameters

addressis the address to which to patch an assembly instruction.

pageidentifies the type of memory to fill: 0 (Program memory), 1 (Data memory) or 2 (I/O space)

For processors that do not have more than one type of memory, use 0 for this Parameters.

patchStringis the assembly string to patch into memory. The simulator allows you to access a data file through a memory address. This function connects a data file to a memory (port) address.

Description

This function loads an assembly string patchString at address on page.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: Yes

Example

GEL_PatchAssembly(0x1000,1, "LAR AR4,#01h");

Note: Patch assembly is not supported for C6000 targets (actual or simulated).

3.8.2.3.42. GEL_PinConnect()

For simulator only, connects a data file to an external interrupt pin.

Syntax

GEL_PinConnect("pinName**", "**fileName**");**

Parameters

pinName names the external interrupt pin. The pinName must be enclosed in quotation marks.

fileNamenames the data file that specifies interrupt intervals. The fileName must be enclosed in quotation marks.

Description

The simulator allows you to simulate external interrupt signals. This function connects a data file that specifies interrupt intervals to the specified external interrupt pin.

If the file is not in the current directory, provide a full path name within the string. A double backslash escape sequence is required to ensure that you get a backslash into the fileName Parameters.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: Yes

Example

GEL_PinConnect("INT0", "c:\\mydir\\myfile.dat");

Related Topics

GEL_PinDisconnect()

3.8.2.3.43. GEL_PinDisconnect()

For simulator only, disconnects a data file from an external interrupt pin.

Syntax

GEL_PinDisconnect("pinName**");**

Parameters

pinNamenames the external interrupt pin. The pinName must be enclosed in quotation marks.

Description

The simulator allows you to simulate external interrupt signals by connecting a data file that specifies interrupt intervals to an external interrupt pin. This function disconnects the currently connected data file from the specified pin.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: Yes

Example

GEL_PinDisconnect("INT0");

Related Topics

GEL_PinConnect()

3.8.2.3.44. GEL_PortConnect()

For simulator only, connects a data file to a memory (port) address.

Syntax

GEL_PortConnect(portAddress**,** page**,** length**,** accessType**, "**fileName**");**

Parameters

portAddressstarting address of a range in memory. This Parameters can be an absolute address, any C expression, the name of a C function, or an assembly language label. If you want to specify a hex address, be sure to prefix the address number with 0x; otherwise, Code Composer Studio treats the number as a decimal address.

pageidentifies the type of memory: 0 (Program memory), 1 (Data memory) or 2 (I/O space)

For processors that do not have more than one type of memory, use 0 for this Parameters.

lengthdefines the length of the range. This Parameters can be any C expression.

accessTypespecifies the type of access allowed. The access type can be any combination of the following values:

Access Type	Value
#define GTI_ATTR_PORT_READ	0x01
#define GTI_ATTR_PORT_WRITE	0x02
#define GTI_ATTR_PORT_NOREWIND	0x04
#define GTI_ATTR_PORT_UPDATE	0x08
#define GTI_ATTR_PORT_EXTERN	0x10

fileNamenames the data file. The fileName must be enclosed in quotation marks.

Description

The simulator allows you to access a data file through a memory address. This function connects a data file to a memory (port) address. You can then read data from the file or write data to the file.

If the file is not in the current directory, provide a full path name within the string. A double backslash escape sequence is required to ensure that you get a backslash into the fileName Parameters.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: Yes

Example

GEL_PortConnect(0x1000, 0, 0x100, , "c:\\mydir\\myfile.dat");

Related Topics

GEL_PortDisconnect()

Connecting a File to a Memory Address

3.8.2.3.45. GEL_PortDisconnect()

For simulator only, disconnects a data file from a memory (port) address.

Syntax

GEL_PinDisconnect(portAddress**,** page**,** length**,** accessType**);**

Parameters

portAddressspecifies the starting address of a range in memory. This Parameters can be an absolute address, any C expression, the name of a C function, or an assembly language label. If you want to specify a hex address, be sure to prefix the address number with 0x; otherwise, Code Composer Studio treats the number as a decimal address.

pageidentifies the type of memory: 0 (Program memory), 1 (Data memory) or 2 (I/O space)

For processors that do not have more than one type of memory, use 0 for this Parameters.

lengthdefines the length of the range. This Parameters can be any C expression.

accessTypespecifies the type of access allowed. The access type can be any combination of the following values:

Access Type	**Value**
#define GTI_ATTR_PORT_READ	0x01
#define GTI_ATTR_PORT_NOREWIND	0x02
#define GTI_ATTR_PORT_UPDATE	0x04
#define GTI_ATTR_PORT_EXTERN	0x08
	0x10

Description

The simulator allows you to read or write data by connecting a data file to a memory (port) address. This function disconnects the currently connected data file from the specified memory address.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: Yes

Example

GEL_PortDisconnect(0x1000, 0, 0x100,);

Related Topics

GEL_PortConnect()

Connecting a File to a Memory Address

3.8.2.3.46. GEL_ProbePtAdd()

Adds a software breakpoint at the specified address.

Syntax

GEL_ProbePtAdd(address**);**

Parameters

addressspecifies the location of the breakpoint. If address is a string Parameters, a symbolic breakpoint will be set at that location and re-evaluated whenever symbols change. Otherwise, a breakpoint will be set at the specified address.

Description

This function sets a software breakpoint at a specified address. The address can be any GEL expression that evaluates to a program address, or a string that specifies a symbolic location. The breakpoint must be manually connected to a window or a file I/O operation before it will do anything.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: No

Example

GEL_ProbePtAdd(PC + 100);

GEL_ProbePtAdd(0x2000);

GEL_ProbePtAdd("main");

Related Topics

GEL_ProbePtDel

GEL_ProbePtReset

GEL_ProbePtDisable

3.8.2.3.47. GEL_ProbePtDel()

Deletes a breakpoint.

Syntax

GEL_ProbePtDel(address**);**

Parameters

addressspecifies the address where a breakpoint will be deleted.

Description

This function allows you to delete a breakpoint at a specified address, if the breakpoint exists.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: No

Example

GEL_ProbePtDel(100);

Related Topics

GEL_ProbePtAdd

GEL_ProbePtReset

3.8.2.3.48. GEL_ProbePtDisable

Disables a breakpoint at the specified address.

Format

GEL_ProbePtDisable(address);

Parameters

addressspecifies the location of the breakpoint.

Description

The breakpoint at the specified address is disabled. If the breakpoint does not exist or is already disabled, no action is taken.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: No

Example

GEL_ProbePtDisable(0x2000);

GEL_ProbePtDisable(main + 100);

Related Topics

GEL_ProbePtAdd

GEL_ProbePtDel

GEL_ProbePtDisable

3.8.2.3.49. GEL_ProbePtReset()

Deletes all breakpoints.

Syntax

GEL_ProbePtReset();

Parameters

None.

Description

This function can be used to remove all currently set breakpoints.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: No

Example

GEL_ProbePtReset();

Related Topics

GEL_ProbePtAdd

GEL_ProbePtDel

3.8.2.3.50. GEL_RefreshWindows

Forces a refresh of all IDE windows

Format

GEL_RefreshWindows();

Parameters

None

Description

This function causes the Disassembly window, Mixed Mode window, Register window, Memory window and the Watch window to refresh their contents with what is on the target. This can be useful if target memory is changing while the target is halted, such as due to a DMA transfer.

Synchronous

Synchronous from GEL: No

Completely synchronous: No

Example

GEL_RefreshWindows

3.8.2.3.51. GEL_RemoveDebugState

Removes the debug state from the target.

Format

GEL_RemoveDebugState();

Parameters

None

Description

This function removes the debug state from the target, much like what happens when the target is disconnected. The debug state includes hardware and software breakpoints, global breakpoints, profiling state, and anything else that might effect target execution. The state can be restored using GEL_RestoreDebugState or selecting Restore Debug State from the Debug menu.

Synchronous

Synchronous from GEL: No

Completely synchronous: No

Example

GEL_RemoveDebugState();

Related Topics

GEL_RestoreDebugState

3.8.2.3.52. GEL_RemoveInputFile()

Removes input file from the breakpoint at the specified address.

Syntax

GEL_RemoveInputFile(programAddr**, "**connectFileName**");**

Removes input file from the breakpoint in the specified file.

Syntax

GEL_RemoveInputFile("srcFileName**",** lineNumber**, "**connectFileName**");**

Parameters

programAddran expression specifying the memory location of the breakpoint to be removed. This expression should not be enclosed in quotation marks.

srcFileNamea string value specifying the filename (without path) of the source file in which the breakpoint to be removed is located. A double backslash escape sequence is required to ensure that a backslash is inserted into the filename; for example: "c:\\ti\\myprojects\\sine.dat".

lineNumbera long integer value specifying the line number in said file at which the breakpoint to be removed is located.

connectFileNamea string value specifying the full path and filename of the file connected with the breakpoint. This connection will be severed and the file closed. A double backslash escape sequence is required to ensure that a backslash is inserted into the filename; for example: "c:\\ti\\myprojects\\sine.dat".

Description

This function disconnects the specified file from the appropriate breakpoint, puts the corresponding input FileI/O object into the Stopped state, closes the input file, and removes the breakpoint (whether or not it was originally added by a call to GEL_AddInputFile).

Synchronous

Synchronous from GEL: Yes

Completely synchronous: No

Related Topics

GEL_RemoveOutputFile

GEL_AddInputFile

3.8.2.3.53. GEL_RemoveOutputFile()

Removes the output file from the breakpoint at the specified address.

Syntax

GEL_RemoveOutputFile(programAddr**, "**connectFileName**");**

Removes the output file from the breakpoint in the specified file.

Syntax

GEL_RemoveOutputFile("srcFileName**",** lineNumber**, "**connectFileName**");**

Parameters

programAddr: is an expression specifying the memory location of the breakpoint to be removed. This expression should not be enclosed in quotation marks.

srcFileName: is a string value specifying the filename (without path) of the source file in which the breakpoint to be removed is located. A double backslash escape sequence is required to ensure that a backslash is inserted into the filename; for example: "c:\\ti\\myprojects\\sine.dat"

lineNumber: is a long integer value specifying the line number in said file at which the breakpoint to be removed is located.

connectFileName: is a string value specifying the full path and filename of the file connected with the breakpoint. This connection will be severed and the file closed. A double backslash escape sequence is required to ensure that a backslash is inserted into the filename; for example: "c:\\ti\\myprojects\\sine.dat"

Description

This function disconnects the specified file from the appropriate breakpoint, puts the corresponding output FileI/O object into the Stopped state, closes the output file, and removes the breakpoint (whether or not it was originally added by a call to GEL_AddOutputFile).

Synchronous

Synchronous from GEL: Yes

Completely synchronous: No

Related Topics

GEL_RemoveInputFile

GEL_AddOutputFile

3.8.2.3.54. GEL_Reset()

Resets target system.

Syntax

GEL_Reset();

Parameters

None.

Description

The Reset() function resets the target system and reloads the monitor. Note that this is a software reset. The GEL_Reset() function performs a device reset via emulation. Peripherals, by design, may or may not be affected by this function.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: Yes

Example

GEL_Reset();

Related Topics

GEL_Restart

GEL_Run

GEL_Halt

3.8.2.3.55. GEL_Restart()

Resets the PC to the program entry point.

Syntax

GEL_Restart();

Parameters

None.

Description

The GEL_Restart() function resets the program to its entry point. This assumes that a program (with symbol information) has been loaded into the target and the target program is able to be restarted. If the target program relies on the loader to initialize various RAM locations, the program may not function correctly after being restarted.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: Yes

Example

GEL_Reset();

Related Topics

GEL_Reset

GEL_Run

GEL_Halt

3.8.2.3.56. GEL_RestoreDebugState

Restores the debug state that was most recently removed from the target.

Format

GEL_RestoreDebugState();

Parameters

None

Description

This function has the same behavior as the Restore Debug State option in the Debug menu. Both restore the debug state to the target that was most recently removed due to a target being disconnected, or by GEL_RemoveDebugState(). The debug state includes hardware and software breakpoints, global breakpoints, profiling state, and anything else that might effect target execution.

Synchronous

Synchronous from GEL: No

Completely synchronous: No

Example

```
GEL_RestoreDebugState();
```

Related Topics

GEL_RemoveDebugState

3.8.2.3.57. GEL_Run()

Runs code.

Syntax

GEL_Run([**"**Condition**"**] **);**

Parameters

Condition: (optional) specifies the condition that must be satisfied while the target is executing. Once the execution reaches a breakpoint and the condition is evaluated to be false, the Code Composer Studio debugger stops at that address. The condition must be enclosed in quotation marks.

Description

This function starts executing code on the target. If a condition is specified, the run function becomes a conditional run statement. That is, execution continues while the statement is true. The statement is evaluated at each breakpoint that is encountered.

Synchronous

Synchronous from GEL: No

Completely synchronous: No

Example

```
GEL_Run();
GEL_Run("A != B");
```

Related Topics

GEL_Restart

GEL_Go

GEL_Halt

GEL_RunF

3.8.2.3.58. GEL_RunF()

Runs free.

Syntax

GEL_RunF();

Parameters

None.

Description

This function disables breakpoints before it starts executing code on the target. It also disconnects from the target system. This is useful if you need to perform a hardware reset on your target system or if you need to disconnect the JTAG or MPSD cable. The Code Composer Studio debugger reconnects to the target system and enables breakpoints if any access is requested on the target system (e.g., memory read), or if the user halts the processor.

Synchronous

Synchronous from GEL: No

Completely synchronous: No

Example

GEL_RunF();

Related Topics

GEL_Restart

GEL_Go

GEL_Halt

GEL_Run

3.8.2.3.59. GEL_SetBlockResetMode()

Syntax

GEL_SetBlockResetMode("integer"**);**

Parameters

integer:

0 [default] - Do not block any reset.

1 - Block reset once.

Description

This GEL function is a part of the emulation wakeup sequence. It halts a target when the target’s power domain is off.

Synchronous

Synchronous from GEL: No

Completely Synchronous: No

Related Topics

GEL_SystemRestoreState

GEL_SystemSaveAllMemory

GEL_SystemSaveAllRegisters

GEL_SystemSaveMemory

GEL_SystemSaveRegister

3.8.2.3.60. GEL_SetSimMode()

Sets the simulator mode.

Syntax

GEL_SetSimMode(mode **);**

Parameters

mode specifies the simulator mode.

Mode	Value
Emulation mode	0
Simulation mode	1 (default)

Description

This function enables you to switch the simulator between emulation and simulation mode. Simulator mode is set by default.

Emulation mode causes the simulator to behave like the emulator with regards to flushing the pipeline when single-stepping. This mode is useful when you want to functionally debug your program.

Use simulation mode when you want to optimize your program by removing pipeline stalls and conflicts. Simulation mode allows you to advance the pipeline by 1 cycle at a time, or by 1 instruction. In simulation mode, the pipeline is not flushed, so it is more cycle accurate.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: No

Example

GEL_SetSimMode(0);

3.8.2.3.61. GEL_SetTimer

Scheduals a gel command to run periodically.

Format

GEL_SetTimer(milliseconds**,** timer_id**, "**callback**");**

Parameters

millisecondsspecifies the amount of time in milliseconds that should elapse between callbacks.

timer_idspecifies a number to uniquely identify this timer.

callbackis a string representing the GEL expression to evaluate every time the timer fires.

Description

Sets a timer that repeatedly fires every milliseconds. Every time the timer fires, the GEL expression defined in callback is executed. The timer keeps firing until GEL_CancelTimer is called. If a timer is already set with the specified id, no action is taken.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: Yes

Example

GEL_SetTimer(2000, 1, "*0x8010 = 0");

GEL_SetTimer(5000, 2, "MyCallback()");

Related Topics

GEL_CancelTimer

3.8.2.3.62. GEL_SetWaitInResetMode()

Sets or clears the Halt on Reset option.

Syntax

GEL_SetWaitInResetMode("integer"**);**

Parameters

integer:

0 - Will not halt on a reset.

1 - Halts at the beginning of the reset vector once a reset has been set.

Description

This GEL function sets or clears the Halt on Reset menu option, which is used for resets that occur externally to the debugger.

Synchronous

Synchronous from GEL: Yes

Completely Synchronous: Yes

Related Topics

GEL_Reset

GEL_Restart

GEL_Run

GEL_Halt

3.8.2.3.63. GEL_SharedMemHaltOnStepOff()

Affected cores are not halted when a breakpoint in shared memory is stepped over.

Syntax

GEL_SharedMemHaltOnStepOff(void **);**

Parameters

None

Description

Multiple cores on a single processor can share a block of memory. When stepping over a software breakpoint that is set in a shared memory block, the breakpoint is first cleared, then the code is stepped, and finally the breakpoint is reset. If other cores execute code in that memory block during this time, they will miss the breakpoint. This function specifies that cores with executable code on that block are not halted.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: Yes

Example

GEL_SharedMemHaltOnStepOff();

Related Topics

GEL_SharedMemHaltOnStepOn

GEL_SharedMemHaltOnWriteOff

GEL_SharedMemHaltOnWriteOn

3.8.2.3.64. GEL_SharedMemHaltOnStepOn

Affected cores are halted when a breakpoint in shared memory is stepped over.

Syntax

GEL_SharedMemHaltOnStepOn(void);

Parameters

None

Description

Multiple cores on a single processor can share a block of memory. When stepping over a software breakpoint that is set in a shared memory block, the breakpoint is first cleared, then the code is stepped, and finally the breakpoint is reset. If other cores execute code in that memory block during this time, they will miss the breakpoint. This function specifies that all cores with executable code on that block are halted until the breakpoint has been stepped over.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: Yes

Example

GEL_SharedMemHaltOnStepOn();

Related Topics

GEL_SharedMemHaltOnStepOff

GEL_SharedMemHaltOnWriteOff

GEL_SharedMemHaltOnWriteOn

3.8.2.3.65. GEL_SharedMemHaltOnWriteOff()

Affected cores are not halted during a write to shared memory.

Syntax

GEL_SharedMemHaltOnWriteOff(void **);**

Parameters

None

Description

Multiple cores on a single processor can share a block of memory. While writing to memory, the breakpoints in that block have to be cleared for a short time by the driver. This function specifies that other cores with executable code on that block are not halted.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: Yes

Example

GEL_SharedMemHaltOnWriteOff();

Related Topics

GEL_SharedMemHaltOnWriteOn

GEL_SharedMemHaltOnStepOff

GEL_SharedMemHaltOnStepOn

3.8.2.3.66. GEL_SharedMemHaltOnWriteOn()

Affected cores are halted during a write to shared memory.

Syntax

GEL_SharedMemHaltOnWriteOn(void **);**

Parameters

None

Description

Multiple cores on a single processor can share a block of memory. While writing to memory, the breakpoints in that block have to be cleared for a short time by the driver. This function specifies that all cores with executable code on that block are halted to ensure that no breakpoints are missed and invalid code is not executed.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: Yes

Example

GEL_SharedMemHaltOnWriteOn();

Related Topics

GEL_SharedMemHaltOnWriteOff

GEL_SharedMemHaltOnStepOff

GEL_SharedMemHaltOnStepOn

3.8.2.3.67. GEL_SrcStepInto()

Steps into source code.

Syntax

GEL_SrcStepInto()

Parameters

count(optional) specifies the number of times the command is to be repeated. If no count is specified, executes a single assembly instruction.

Description

Executes the next available instruction in C or assembly source code then halts. If an action enters an area with no source code, the debugger continues stepping until source code (somewhere) is reached. Repeats count times.

Synchronous

Synchronous from GEL: No

Completely synchronous: No

Related Topics

GEL_AsmStepInto

GEL_AsmStepOver

GEL_SrcStepOver

3.8.2.3.68. GEL_SrcStepOver()

Step over source code.

Syntax

GEL_SrcStepOver()

Parameters

count(optional) specifies the number of times the command is to be repeated. If no count is specified, executes a single assembly instruction.

Description

Execute the next available instruction in C or assembly source without following function calls then halt. If an action brings you into an area with no source code, the debugger continues stepping until source code (somewhere) is reached. Repeats count times.

The source file can be viewed entirely in C or displayed at the same time as the assembly instructions. In C source mode, this command steps over an entire C instruction. Otherwise, it steps over a single assembly instruction. However, to protect the processor's pipeline, several instructions following a delayed branch or call may be considered part of the same statement. In this case, the command may execute more than one instruction at a time.

Synchronous

Synchronous from GEL: No

Completely synchronous: No

Related Topics

GEL_AsmStepInto

GEL_AsmStepOver

GEL_SrcStepInto

3.8.2.3.69. GEL_StepInto

Executes the next statement.

Format

GEL_StepInto();

Parameters

None

Description

The debugger executes the next statement, then halts. If the next statement is a function call, the debugger steps into the function, and then halts execution at the beginning of the function. If you are in C source mode, this command steps through a single C instruction; otherwise, it steps through a single assembly instruction.

Synchronous

Synchronous from GEL: No

Completely synchronous: No

Example

GEL_StepInto( );

Related Topics

GEL_StepOut

GEL_StepOver

GEL_SyncHalt

GEL_SyncStepInto

GEL_SyncStepOver

GEL_SyncStepOut

3.8.2.3.70. GEL_StepOut

Executes a subroutine then halts upon return to the calling function.

Format

GEL_StepOut();

Parameters

None

Description

The debugger executes the current subroutine and then returns to the calling function. Execution halts after returning to the calling function.

A step out can run to an unexpected point if the correct calling function can’t be determined. This can happen in some assembly functions that use the stack to store information or if the stack pointer is incorrect.

A step out is not allowed when in a function that is at the entry point of the program, or when in a function that has minimal debug information.

Synchronous

Synchronous from GEL: No

Completely synchronous: No

Example

GEL_StepOut();

Related Topics

GEL_ClearProfileConfiguration

GEL_StepInto

GEL_StepOver

GEL_SyncHalt

GEL_SyncStepInto

GEL_SyncStepOut

GEL_SyncStepOver

3.8.2.3.71. GEL_StepOver

Executes a function or the next statement.

Format

GEL_StepOver();

Parameters

None

Description

The debugger executes the function and then halts after the function returns. If a breakpoint is encountered within the function, execution halts at the breakpoint.

The Step Over command can also be used to execute statements that are not function calls. In this case, the debugger executes the next statement then halts.

If you are in C source mode, this command steps over an entire C instruction; otherwise, it steps over a single assembly instruction. However, to protect the processor's pipeline, several instructions following a delayed branch or call may be considered part of the same statement. In this case, the Step Over command may execute more than one instruction at a time.

Synchronous

Synchronous from GEL: No

Completely synchronous: No

Example

GEL_StepOver();

Related Topics

GEL_ClearProfileConfiguration

GEL_StepInto

GEL_StepOut

GEL_SyncHalt

GEL_SyncStepInto

GEL_SyncStepOut

GEL_SyncStepOver

3.8.2.3.72. GEL_SymbolAdd()

Loads additional symbol information.

Syntax

GEL_SymbolAdd("fileName**"** [**, "**cpuName**"**] [**, "**boardName**"**]**);**

Parameters

fileNamenames the symbol file containing the symbol information. The fileName must be enclosed in quotation marks. The symbol file can be a standard COFF output file (*.out) or a specific symbol file obtained during the build process.

cpuName(optional) names the CPU on which to load the symbolic information (useful in a multiprocessor environment). The cpuName must be enclosed in quotation marks.

boardName(optional) names the board on which to load the symbolic information. If not specified, uses the board name of the processor on which this GEL function is executed. The boardName must be enclosed in quotation marks.

Description

Symbol information can be appended to the existing symbol table. This function loads the symbol information from the specified symbol file. GEL_SymbolAdd differs from GEL_SymbolLoad in that it does not clear the contents of the existing symbol table before loading the new symbols.

If the file is not in the current directory, provide a full path name within the string. A double backslash escape sequence is required to ensure that you get a backslash into the fileName Parameters.

The cpuName and boardName Parameters must match the Processor Name and Board Name as configured in the multiprocessor setup. In a single processor system, you do not need to fill these fields.

Synchronous

Synchronous from GEL: No

Completely synchronous: No

Example

GEL_SymbolAdd("c:\\mydir\\myfile.out", "cpu_a", "Emulator");

Related Topics

GEL_Load()

GEL_SymbolAddRel()

GEL_SymbolLoad()

GEL_SymbolAddRel()

GEL_SymbolRemove()

GEL_UnloadAllSymbols()

3.8.2.3.73. GEL_SymbolAddRel()

Loads additional symbol information with relocation information.

Syntax

GEL_SymbolAddRel("fileName**",** code_start**,** data_start[**, "**cpuName**"**[**, "**boardName**"**]] **);**

Parameters

fileNamenames the symbol file containing the symbol information. The fileName must be enclosed in quotation marks. The symbol file can be a standard COFF output file (*.out) or a specific symbol file obtained during the build process.

code_startspecifies the starting address where the code sections are to be loaded in memory.

data_startspecifies the starting address where the data sections are to be loaded in memory.

cpuName(optional) names the CPU on which to load the symbolic information (useful in a multiprocessor environment). The cpuName must be enclosed in quotation marks.

boardName(optioanal) names the board on which to load the symbolic information. If not specified, uses the board name of the processor on which this GEL function is executed. The boardName must be enclosed in quotation marks.

Description

Symbol information can be appended to the existing symbol table. This function loads the symbol information from the specified symbol file. GEL_SymbolAdd and GEL_SymbolAddRel differ from GEL_SymbolLoad and GEL_SymbolLoadRel in that they do not clear the contents of the existing symbol table before loading the new symbols. Symbols are loaded at the code and data addresses specified by the relocation information.

If the file is not in the current directory, provide a full path name within the string. A double backslash escape sequence is required to ensure that you get a backslash into the filename entry.

The relocation information is specified using an absolute address that identifies where the sections are to be loaded in memory.

The cpuName and boardName must match the Processor Name and Board Name as configured in the multiprocessor setup. In a single processor system, you do not need to fill these fields.

Synchronous

Synchronous from GEL: No

Completely synchronous: No

Example

GEL_SymbolAddRel("c:\\mydir\\myfile.out", 16, 64, "cpu_a", "Simulator");

Related Topics

GEL_Load()

GEL_SymbolAdd()

GEL_SymbolLoad()

GEL_SymbolRemove()

GEL_UnloadAllSymbols()

3.8.2.3.74. GEL_SymbolHideSection()

Hides a section of symbols.

Syntax

GEL_SymbolHideSection("fileName**", "**sectionName**"** [**, "**cpuName**"** [**, "**boardName**"**]] **);**

Parameters

fileNamenames the symbol file containing the symbol information. The fileName must be enclosed in quotation marks. The symbol file can be a standard COFF output file (*.out) or a specific symbol file obtained during the build process.

sectionNamenames the section to remove. The sectionName must be enclosed in quotation marks.

cpuName(optional) names the CPU on which to load the symbolic information (useful in a multiprocessor environment). The cpuName must be enclosed in quotation marks.

boardName(optional) names the board on which to load the symbolic information. If not specified, uses the board name of the processor on which this GEL function is executed. The boardName must be enclosed in quotation marks.

Description

A section of symbols can be hidden. The symbols are still loaded but cannot be seen by the debugger. To show them again you must use GEL_SymbolShowSection().

Synchronous

Synchronous from GEL: No

Completely synchronous: No

Example

GEL_SymbolHideSection("C:\\mydir\\myfile.out", ".text", "CPU", "C55x Simulator (Texas Instruments)");

Related Topics

GEL_Load()

GEL_SymbolAdd()

GEL_SymbolAddRel()

GEL_SymbolLoad()

GEL_SymbolLoadRel()

GEL_SymbolAddRel()

GEL_SymbolRemove()

GEL_SymbolShowSection()

GEL_UnloadAllSymbols()

3.8.2.3.75. GEL_SymbolLoad()

Loads symbol information only.

Syntax

GEL_SymbolLoad("fileName**"** [**, "**cpuName**"** [**, "**boardName**"**]] **);**

Parameters

fileNamenames the symbol file containing the symbol information. The fileName must be enclosed in quotation marks. The symbol file can be a standard COFF output file (*.out) or a specific symbol file obtained during the build process.

cpuName(optional) names the CPU on which to load the symbolic information (useful in a multiprocessor environment). The cpuName must be enclosed in quotation marks.

boardName(optional) names the board on which to load the symbolic information. If not specified, uses the board name of the processor on which this GEL function is executed. The boardName must be enclosed in quotation marks.

Description

It is useful to load only symbol information when working in a debugging environment where the debugger cannot or need not load the object code, such as when the code is in ROM. This function loads the symbol information from the specified symbol file.

The debugger first deletes any previously loaded symbols from the symbol table maintained on the host. The symbols in the symbol file are then loaded into the symbol table. Symbols are loaded at the code and data addresses specified in the symbol file. This command does not modify memory or set the program entry point.

If the file is not in the current directory, provide a full path name within the string. A double backslash escape sequence is required to ensure that you get a backslash into the fileName entry.

The cpuName and boardName must match the Processor Name and Board Name as configured in the multiprocessor setup. In a single processor system, you do not need to fill these fields.

Synchronous

Synchronous from GEL: No

Completely synchronous: No

Example

GEL_SymbolLoad("c:\\mydir\\myfile.out", "cpu_a", "Emulator");

Related Topics

GEL_Load()

GEL_SymbolAdd()

GEL_SymbolAddRel()

GEL_SymbolLoadRel()

GEL_SymbolAddRel()

GEL_SymbolRemove()

GEL_SymbolShowSection()

GEL_UnloadAllSymbols()

3.8.2.3.76. GEL_SymbolLoadRel()

Loads symbol information with relocation information.

Syntax

GEL_SymbolLoadRel("fileName**",** code_start**,** data_start [**, "**cpuName**"**[**, "**boardName**"**]] **);**

Parameters

fileNamenames the symbol file containing the symbol information. The fileName must be enclosed in quotation marks. The symbol file can be a standard COFF output file (*.out) or a specific symbol file obtained during the build process.

code_startspecifies the starting address where the code sections are to be loaded in memory.

data_startspeicfifes the starting address where the data sections are to be loaded in memory.

cpuName(optional) names the CPU on which to load the symbolic information (useful in a multiprocessor environment). The cpuName must be enclosed in quotation marks.

boardName(optional) names the board on which to load the symbolic information. If not specified, uses the board name of the processor on which this GEL function is executed. The boardName must be enclosed in quotation marks.

Description

It is useful to load only symbol information when working in a debugging environment where the debugger cannot or need not load the object code, such as when the code is in ROM. This function loads the symbol information from the specified symbol file.

The debugger first deletes any previously loaded symbols from the symbol table maintained on the host. The symbols in the symbol file are then loaded into the symbol table. Symbols are loaded at the code and data addresses specified by the relocation information. This command does not modify memory or set the program entry point.

If the file is not in the current directory, provide a full path name within the string. A double backslash escape sequence is required to ensure that you get a backslash into the fileName entry.

The relocation information is specified using an absolute address that identifies where the sections are to be loaded in memory.

The cpuName and boardName must match the Processor Name and Board Name as configured in the multiprocessor setup. In a single processor system, you do not need to fill these fields.

Synchronous

Synchronous from GEL: No

Completely synchronous: No

Example

GEL_SymbolLoadRel("c:\\mydir\\myfile.out", 16, 64, "cpu_a", "Simulator");

Related Topics

GEL_Load()

GEL_SymbolAdd()

GEL_SymbolAddRel()

GEL_SymbolLoad()

GEL_SymbolLoadRel()

GEL_SymbolAddRel()

GEL_SymbolRemove()

GEL_SymbolShowSection()

GEL_UnloadAllSymbols()

3.8.2.3.77. GEL_SymbolRemove()

Removes symbol information.

Syntax

GEL_SymbolRemove("fileName**"** [**, "**cpuName**"** [**, "**boardName**"**]] **);**

Parameters

fileNamenames the symbol file containing the symbol information. The fileName must be enclosed in quotation marks. The symbol file can be a standard COFF output file (*.out) or a specific symbol file obtained during the build process.

cpuName(optional) names the CPU from which to remove the symbolic information (useful in a multiprocessor environment). The cpuName must be enclosed in quotation marks.

boardName(optional) names the board from which to remove the symbolic information. If not specified, uses the board name of the processor on which this GEL function is executed. The boardName must be enclosed in quotation marks.

Description

This function removes the loaded symbols that are defined in the specified symbol file.

If the file is not in the current directory, provide a full path name within the string. A double backslash escape sequence is required to ensure that you get a backslash into the fileName entry.

The cpuName and boardName must match the Processor Name and Board Name as configured in the multiprocessor setup. In a single processor system, you do not need to fill these fields.

Synchronous

Synchronous from GEL: No

Completely synchronous: No

Example

GEL_SymbolRemove("c:\\mydir\\myfile.out", "cpu_a", "Emulator");

Related Topics

GEL_Load()

GEL_SymbolAdd()

GEL_SymbolAddRel()

GEL_SymbolLoad()

GEL_SymbolLoadRel()

GEL_SymbolAddRel()

GEL_SymbolShowSection()

GEL_UnloadAllSymbols()

3.8.2.3.78. GEL_SymbolShowSection()

Shows a section of symbols.

Syntax

GEL_SymbolShowSection("fileName**", "**sectionName**",** codeStart**,** dataStart [**, "**cpuName**"**[**, "**boardName**"**]] **);**

Parameters

fileNamenames the symbol file containing the symbol information. The fileName must be enclosed in quotation marks. The symbol file can be a standard COFF output file (*.out) or a specific symbol file obtained during the build process.

sectionNamenames the section to remove. The sectionName must be enclosed in quotation marks.

code_startspecifies the starting address where the code sections are to be loaded in memory.

data_startspecifies the starting address where the data sections are to be loaded in memory.

cpuName(optional) names the CPU on which to load the symbolic information (useful in a multiprocessor environment). The cpuName must be enclosed in quotation marks.

boardName(optional) names the board on which to load the symbolic information. If not specified, uses the board name of the processor on which this GEL function is executed. The boardName must be enclosed in quotation marks.

Description

This function shows a section of symbols. It may be useful to show symbol information from specific sections only. If sectionName is a code section, the section will be relocated to address codeStart. If sectionName is a data section, the section will be relocated to address dataStart. If a section is moved, symbols in the section will be relocated accordingly.

Synchronous

Synchronous from GEL: No

Completely synchronous: No

Example

GEL_SymbolShowSection("C:\\mydir\\myfile.out", ".text", 0x0, 0x0, "CPU", "C55x Simulator (Texas Instruments)");

Related Topics

GEL_Load()

GEL_SymbolAdd()

GEL_SymbolAddRel()

GEL_SymbolHideSection()

GEL_SymbolLoad()

GEL_SymbolLoadRel()

GEL_SymbolAddRel()

GEL_SymbolRemove()

GEL_UnloadAllSymbols()

3.8.2.3.79. GEL_SyncHalt

Perform a halt operation on every target that is in the same group as the current target.

Format

GEL_SyncHalt( );

Parameters

None

Description

Performs a halt operation on every target that is in the same group as the current target. The operation performed is the same as the halt command that may be invoked from the PDM debug menu.

As with the other synchronous operations, the operation is performed on all targets simultaneously.

Synchronous

Synchronous from GEL: No

Completely synchronous: No

Example

GEL_SyncHalt( );

Related Topics

GEL_SyncRun

GEL_SyncStepInto

GEL_SyncStepOut

GEL_SyncStepOver

3.8.2.3.80. GEL_SyncRun

Performs a run operation on every target that is in the same group as the current target.

Format

GEL_SyncRun( );

Parameters

None.

Description

Performs a run operation on every target that is in the same group as the current target. The operation performed is the same as the run command that may be invoked from the PDM debug menu.

As with the other synchronous operations, the operation is performed on all targets simultaneously.

Synchronous

Synchronous from GEL: No

Completely synchronous: No

Example

GEL_SyncRun( );

Related Topics

GEL_SyncHalt

GEL_SyncStepInto

GEL_SyncStepOut

GEL_SyncStepOver

3.8.2.3.81. GEL_SyncStepInto

Performs a StepInto operation on every target that is in the same group as the current target.

Format

GEL_SyncStepInto([iterations]**);**

Parameters

iterations(optional) specifies the number of times the command is to be repeated. If iterations is not specified, executes a single assembly instruction.

Description

Performs a StepInto operation on every target that is in the same group as the current target. The operation performed is the same as the Lock Step option selected from the PDM debug menu. The step operation is context sensitive. If stepping through source and source symbolic information is available, a source step-into will be performed. If source symbolic information is not available, an assembly step into will be performed.

As with the other synchronous operations, the operation is performed on all targets simultaneously.

Synchronous

Synchronous from GEL: No

Completely synchronous: No

Example

GEL_SyncStepInto(10);

GEL_SyncStepInto( );

Related Topics

GEL_SyncStepOver

GEL_SyncStepOut

GEL_SyncRun

GEL_SyncHalt

3.8.2.3.82. GEL_SyncStepOut

Performs a StepOut operation on every target that is in the same group as the current target.

Format

GEL_SyncStepOut([iterations]**);**

Parameters

iterations(optional) specifies the number of times the command is to be repeated. If iterations is not specified, executes a single StepOut is performed.

Description

Performs a StepOut operation on every target that is in the same group as the current target. The operation performed is the same as the "StepOut" option selected from the PDM debug menu.

As with the other synchronous operations, the operation is performed on all targets simultaneously.

Synchronous

Synchronous from GEL: No

Completely synchronous: No

Example

GEL_SyncStepOut (3);

GEL_SyncStepOut ( );

Related Topics

GEL_SyncStepInto

GEL_SyncStepOver

GEL_SyncRun

GEL_SyncHalt

3.8.2.3.83. GEL_SyncStepOver

Performs a StepOver operation on every target that is in the same group as the current target.

Format

GEL_SyncStepOver([iterations**]);**

Parameters

iterations(optional) specifies the number of times the command is to be repeated. If iterations is not specified, a single StepOver is performed.

Description

Performs a StepOver operation on every target that is in the same group as the current target. The operation performed is the same as the StepOver option selected from the PDM debug menu.

As with the other synchronous operations, the operation is performed on all targets simultaneously.

Synchronous

Synchronous from GEL: No

Completely synchronous: No

Example

GEL_SyncStepOver(5);

GEL_SyncStepOver ( );

Related Topics

GEL_SyncStepOut

GEL_SyncStepInto

GEL_SyncRun

GEL_SyncHalt

3.8.2.3.84. GEL_System()

Executes a DOS command.

Syntax

GEL_System("dosCommand**"** [**,** param1**,** param2**,** .. param4] **);**

Parameters

dosCommand: specifies the DOS command (which may contain optional format specifiers) that is to be executed.

param1..param4(optional) name additional parameters that are substituted in the dosCommand when a format specifier is encountered. These parameters allow you to pass values to the DOS command.

Description

The GEL_System function allows you to execute DOS commands from within the IDE. The output of the DOS command is sent to an output window within the IDE. The DOS command can only be one that produces a text output and does not require additional user input once it starts executing.

The DOS command that is executed is actually the formatted string given by dosCommand and the additional parameters (parm1..parm4). This allows you to pass additional parameters (including values that are defined on the target) to the DOS command.

Format specifications always begin with a percent sign (%) and are read left to right. When the first format specification (if any) is encountered, the value of the first argument after format is converted and printed in dosCommand. The second format specification causes the second argument to be converted and printed, and so on. If there are more arguments than there are format specifications, the extra arguments are ignored.

The GEL_System() GEL function is implemented using proprietary technology and can be used to extend the capabilities of the IDE. You can use it to perform tasks (such as compiling) in the background and pipe the results to the output window within the IDE.

Synchronous

Synchronous from GEL: No

Completely synchronous: No

Example

GEL_System("dir");
GEL_System("dir *.dat")

Which is equivalent to:

GEL_System("dir %s", "*.dat");

But not GEL_System("dir", "*.dat");

GEL_System("myfunc %f %d %s", targVar, 3, "-ol");

If we assume that targVar is a variable defined on the target and that its value is 3.14, then the final DOS command that is executed is: >>myfunc 3.14 3 -ol

3.8.2.3.85. GEL_TargetTextOut()

Display Target Formatted String.

Syntax

GEL_TargetTextOut(startAddress [**,** page [**,** maxLength [**,** Format [**, "**outputLabel**"** [**,** textColor [**,** lineNumber [**,** appendToEnd [, changeHighlight]]]]]]] **);**

Parameters

startAddressspecifies the first address of the block containing the preformatted string.

page(optional) identifies the type of memory: 0 (Program memory), 1 (Data memory) or 2 (I/O space)

For processors that do not have more than one type of memory, use 0 for page. The default value for the page number is 0. For simulated targets, the I/O space parameters is not supported.

maxLength(optional) specifies the maximum length of the block if the block is longer than 400 bytes. The formatted string on the target should be a null terminated string. However, if a null is not encountered only 400 or maxLength bytes (whichever is larger) of the string will be printed.

Format(optional) indicates whether the text is in packed or unpacked format on the target:

0	ASCII character (per addressable unit)
1	Packed ASCII character using big endian format; the first character is in the most significant byte of the target.
2	ASCII character using little endian format; the first character is in the least significant byte of the target.

outputLabel(optional) names the label that GEL prepends in its output. The outputLabel must be enclosed in quotation marks. By default, "GEL Output" is used.

The remaining arguments are ignored in this version of CCS.

Description

This function is used to print a formatted string to the console window. The string must already exist on the target and must be null terminated.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: Yes

Example

```
GEL_TargetTextOut(0x800);
GEL_TargetTextOut(0x1000, 0, 400, 1, "My Window", 1);
```

Related Topics

GEL_TextOut()

GEL_EnableFileOutput

3.8.2.3.86. GEL_TextOut()

Prints text to the console view.

Syntax

GEL_TextOut("text**"** [**, "**outputLabel**"**[**,** textColor[, lineNumber[**,** appendToEnd [**,** param1**,** param2**,** .. param4]]]] **);**

Parameters

text is the formatted text (including format specifiers) that is to be printed. The number of format specifiers must match the number of additional parameters (param1.. param4) that are encountered.

outputLabel(optional) names the label that GEL prepends in its output. The outputLabel must be enclosed in quotation marks. By default, "GEL Output" is used.

The arguments textColor, lineNumber and appendToEnd are ignored in this version of CCS.

param1..param4(optional) additional parameters that are substituted in text when a format specifier is encountered. These parameters allow you to pass values to the output window.

Description

This function prints a fixed string to a specified output window. This function is ideal for printing messages.

Format specifications always begin with a percent sign (%) and are read left to right. When the first format specification (if any) is encountered, the value of the first argument (parm1) is converted and printed in an output window within the IDE. The second format specification causes the second argument to be converted and printed, and so on. If there are more arguments than there are format specifications, the extra arguments are ignored.

Note: the function will have no effect when used in the StartUp() function because it requires that the IDE control window be open beforehand and StartUp() isn't active when the control window is open.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: Yes

Example

```
GEL_TextOut("All Tests Passed\n");
GEL_TextOut("Failed Memory Test\n", "Diagnostic Results", 2 );
GEL_TextOut("Tests Executed: %d, Tests Passed %d ",,,,,targExe, targPass);
```

Related Topics

GEL_TargetTextOut

GEL_EnableFileOutput

3.8.2.3.87. GEL_UnloadAllSymbols()

Unloads all symbols.

Syntax

GEL_UnloadAllSymbols();

Parameters

None.

Description

Unloads all currently loaded symbols.

Synchronous

Synchronous from GEL: No

Completely synchronous: No

Example

```
GEL_UnloadAllSymbols();
```

Related Topics

GEL_Load()

GEL_SymbolAdd()

GEL_SymbolAddRel()

GEL_SymbolHideSection()

GEL_SymbolLoad()

GEL_SymbolLoadRel()

GEL_SymbolAddRel()

GEL_SymbolRemove()

GEL_UnloadAllSymbols()

3.8.2.3.88. GEL_UnloadGEL()

Unloads a GEL file.

Syntax

GEL_UnloadGel("fileName**");**

Parameters

fileNamenames the GEL file to be unloaded. The fileName must be enclosed in quotation marks.

Description

This function unloads the functions defined in the specified GEL file.

The fileName Parameters is case sensitive. The filename must be specified exactly as it was when the file was loaded. If the file is not in the current directory, the fileName Parameters must provide the full path. A double backslash escape sequence is required to ensure that you get a backslash into the fileName Parameters.

Note: When a file is opened using menu commands, Windows capitalizes the drive letter.

Synchronous

Synchronous from GEL: Yes

Completely synchronous: Yes

Example

GEL_UnloadGel("C:\\MyDir\\myfile.gel");

Related Topics

GEL_LoadGel

GEL_UnloadAllGels

3.8.3. GEL Macros

Now there is a way to make GEL files portable across different machines by using macros.

In GEL files, you can use the macro "$(GEL_file_dir)" in an expression. Before evaluating, GEL will replace that with the directory that the GEL file is located in. This can simplify strings and / or make it easier to share GEL files between users.

Now instead of having to type GEL_Load("C:\\My Documents\\workspace\\My Projects\\Bull\\bull.out"), you can simply type GEL_Load("$(GEL_file_dir)\\bull.out").

Example:

GEL_Load("$(GEL_file_dir)\\bull.out")

3.8.4. Loading and Unloading GEL Files

Startup GEL files defined in the target configuration file will be automatically loaded when a debug session is started.

Additional GEL files can be loaded in the CCS from the 'GEL Files' dialog (via 'Tools->GEL Files' menu).

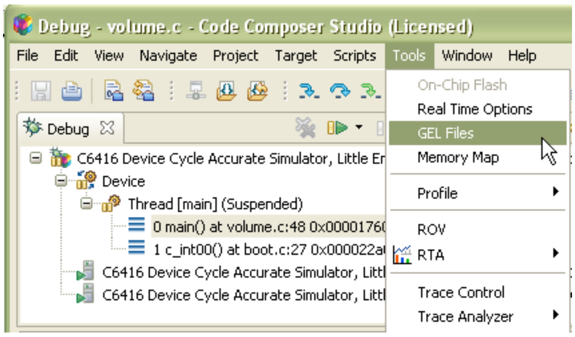

Debug - volume.c - Code Composer Studio (Licensed)
File
Edit
View
Navigate
Project
Target
Scripts
Tools
Window
Help
On-Chip Flash
Real Time Options
GEL Files
Memory Map
Profile
ROV
RTA
Trace Control
Trace Analyzer
Debug
C6416 Device Cycle Accurate Simulator, Little En
Device
Thread [main] (Suspended)
0 main() at volume.c:48 0x0000176
1 c_int00() at boot.c:27 0x000022a
C6416 Device Cycle Accurate Simulator, Littl
C6416 Device Cycle Accurate Simulator, Littl

Fig. 1: 'GEL Files' Menu

This will open the 'GEL Files' view. From this view, users can see all loaded GEL files for that debug session and have the option to open them in the editor, reload the GEL file, remove/unload the GEL file, and load additional GEL files. Right-click in the view and select 'Load GEL' to load a GEL file:

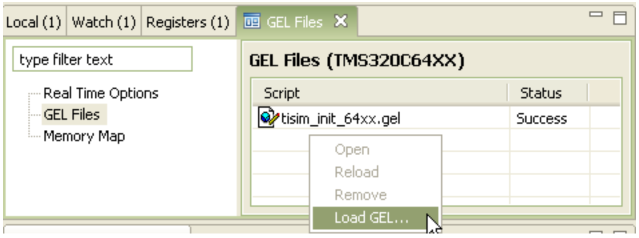


Fig. 2: 'GEL Files' View

When a new GEL file is loaded, it will appear in the 'GEL Files' view. If the load was successful, the status will be listed as 'Success'.

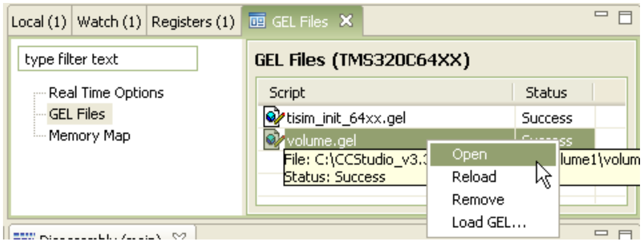


Fig. 3: GEL File Loaded

3.8.5. Adding GEL Functions to the GEL Menu Using Keywords

You can add GEL functions that you access frequently to the GEL menu of the Code Composer Studio menu bar. To do this, use the menuitem keyword to create a new drop-down list of menu items under the GEL menu. You can then use the keywords hotmenu, dialog, or slider to add new menu items in the most recent drop-down list. When you select the user defined menu item (under the GEL menu), a dialog box or slider object appears.

Note: There is no limit to the number of GEL functions that can be defined. However, the number of GEL functions that appear on the GEL menu may be limited by the size of your screen and the resolution.

Related Topics

The GEL hotmenu Keyword

The GEL dialog Keyword

The GEL slider Keyword

3.8.5.1. The hotmenu Keyword

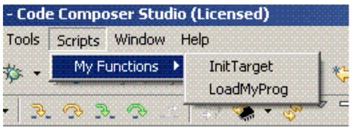
Use the hotmenu keyword to add a GEL function to the GEL menu that is executed immediately when selected. The syntax is as follows:

hotmenu funcName**()**
{
statements
}

This keyword is used for GEL functions that have no Parameters to be passed.

```
menuitem "My Functions";
hotmenu InitTarget()
{
*waitState = 0x11;
}
hotmenu LoadMyProg()
{
GEL_Load("c:\\mydir\\myfile.out");
}
```

This example adds the following sub-selections under the Scripts menu.



When you choose the InitTarget command, it is immediately executed. To call GEL functions that require Parameters to be passed, use the dialog keyword.

3.8.5.2. The dialog Keyword

Use the dialog keyword to add a GEL function to the Scripts menu and to create a dialog window for Parameters entry. When you select the function from the Scripts menu, a dialog window appears to prompt you for the Parameters to enter. The strings beside the Parameters in the function declaration are for Parameters descriptions in the dialog box. The syntax of a dialog GEL function is as follows:

dialog funcName**(** paramName1 **"**param1 definition**",** paramName2 **"**param2 definition**",** ... **)**
{
statements
}

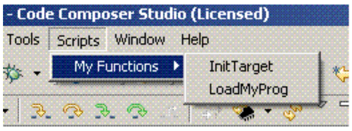
paramName[1-6]Parameters variable name that is used inside the function

"param1 definition"Parameters description that is printed on the dialog window beside the field

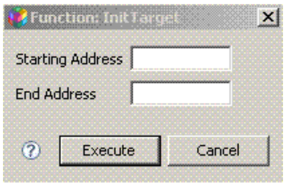
You can pass up to six Parameters to the added GEL function through the dialog window. The following example shows how you can use the dialog keyword to add two menu items.

```
menuitem "My Functions";
dialog InitTarget(startAddress "Starting Address", EndAddress "End Address")
{
statements
}
dialog LoadMyProg()
{
statements
}
```

This example adds the following sub-selections under the Scripts menu.



When you use the InitTarget command, the Function: InitTarget dialog box prompts you for the start and end addresses.



When you enter values into the entry fields, press the Execute button to call the GEL function with these Parameters.

3.8.5.3. The slider Keyword

You can also use the slider keyword to add a GEL function to the Scripts menu. When you select the function from the Scripts menu, a slider object appears to control the value passed to the GEL function. Each time you move the position of the slider, the GEL function is called with a new Parameters value reflecting the new position of the slider. You can only pass one Parameters to a slider GEL function. The Format of a slider GEL function is as follows:

slider param_definition(minVal**,** maxVal**,** increment**,** pageIncrement**,** paramName **)**
{
statements
}

param_definitionParameters description that is printed on the slider object.

minVal An integer constant specifying the value to be passed to the function when the position of the slider is at its lowest level.

maxVal An integer constant specifying the value to be passed to the function when the position of the slider is at its highest level.

increment An integer constant specifying the increment added to the value each time the slider is moved one position.

pageIncrement An integer constant specifying the increment added to the value each time the slider is moved by one page.

paramName Parameters definition that is used inside the function.

The following example uses the slider keyword to add a volume control slider.

```
menuitem "My Functions";
slider VolumeControl(0, 10, 1, 1, volume)
{
/* initialize the target variable with the Parameters passed by the slider object. */
targVarVolume = volume;
}
```

4.1.1. Device Documentation and User Guides

This page contains external links to pages at www.ti.com that provide Application Notes, User Guides, Device information, software and training.

Device Family Specific links.

C2000 Devices.

C5000 Devices.

C6000 Devices.

OMAP Devices.

MSP430 Devices.

Stellaris Devices.

Integra Devices.

Sitara Devices.

Submit Code Composer Documentation Feedback

6.1. Asia Product Information Centers

International	+91-80-41381665	
Domestic		**Toll-Free Number**
	Australia	1-800-999-084
	China	800-820-8682
	Hong Kong	800-96-5941
	India	1-800-425-7888
	Indonesia	001-803-8861-1006
	Korea	080-551-2804
	Malaysia	1-800-80-3973
	New Zealand	0800-446-934
	Philippines	1-800-765-7404
	Singapore	800-886-1028
	Taiwan	0800-006800
	Thailand	001-800-886-0010
Fax	+886-2-2378-6808	
Email	tiasia@ti.com	
	ti-china@ti.com	

Internet	support.ti.com/sc/pic/asia.htm	

For Japan

Fax

- International: +81-3-3344-5317
- Domestic: 0120-81-0036

Internet/Email

- International—support.ti.com/sc/pic/japan.htm
- Domestic—www.tij.co.jp/pic

DSP BBS via Nifty-Serve Type "Go TIASP"

6.2. Documentation Comments

When making suggestions or reporting errors in documentation, please include this information that is on the title page: the full title of the book, the publication date, and the literature number. When mailing to TI:

> Texas Instruments Incorporated
> Technical Documentation Services, MS 702
> P.O. Box 1443
> Houston, Texas 77251-1443

For sending documentation comments by email: support@ti.com

Click this link anywhere in the Code Composer Help to provide specific documentation comments on the Internet.

6.3. Europe, Middle East, Africa

Telephone

European Free Call		00800-ASK-TEXAS (00800 275 83927)
International		+49 (0) 8161 80 2121
	Note:	The European Free Call (Toll Free) is not active in all countries. If you have technical difficulty calling the toll free call number, please use the International number.

Specific Country Telephone Numbers

Belgium (English)	+32 (0) 27 45 54 32
Finland (English)	+358 (0) 9 25173948
France	+33 (0) 1 30 70 11 64
Germany	+49 (0) 8161 80 33 11
Israel (English)	1800 949 0107
Italy	800 79 11 37
Netherlands (English)	+31 (0) 546 87 95 45
Russia	+7 (4) 95 98 10 701
Spain	+34 902 35 40 28
Sweden (English)	+46 (0) 8587 555 22
United Kingdom	+44 (0) 1604 66 33 99

Miscellaneous

Fax	+(49) (0) 8161 80 2045
Product Information Center (Internet)	support.ti.com/sc/pic/euro.htm
EPIC Modem BBS	+33 1 30 70 11 99
European Factory Repair	+33 4 93 22 25 40
Europe Customer Training Helpline (Fax)	+ 49 81 61 80 40 10

6.4. If You Need Assistance - TI Worldwide Technical Support

Refer to these links for worldwide or regional support. When calling a Literature Response Center to order documentation, please specify the literature number of the document.

If you want to download an available document, please begin your search with the Code Composer Documentation links.

Customer support areas include:

- Worldwide Internet Sites
- North America, South America, Central America
- Europe, Middle East, Africa
- Asia-Pacific
- Japan
- Texas Instruments Products and Application Solutions
- TI Product Information Center
- Documentation
- Technical Training

6.5. Internet Sites

Texas Instruments Semiconductor Internet Pages

TI Semiconductor Home Page	www.ti.com
TI Semiconductor Distributors	focus.ti.com/docs/general/distributor.jhtml
Semiconductor Product Information Center (PIC)	support.ti.com
Semiconductor Knowledge Base Home Page	support.ti.com/sc/knowledgebase
My TI homepage	www.ti.com/home_m_allmyti

DSP Specific Internet Pages

DSP Solutions	www.ti.com/dsps
DSP Knowledge Base Home Page	www-k.ext.ti.com/sc/technical-support/dsp-kbase
Framework Software	
Tools and Software	focus.ti.com/dsp/docs/dspfindtoolswbytooltype.tsp?sectionId=3&tabId=2088&toolTypeId=1&familyId=44

To check for updated URLs, go to www.ti.com or www.dspvillage.com. If you still cannot find what you want, try the DSP Knowledge Base (see above), or see How and Where to Look for the Information You Need.

6.6. Japan

Fax		
	International	+81-3-3344-5317
	Domestic	0120-81-0036
Internet		
	International	support.ti.com/sc/pic/japan.htm
	Domestic	www.tij.co.jp/pic
DSP BBS via Nifty-Serve Type	"Go TIASP"	

6.7. Americas Product Information Center

TI Semiconductor Home Page	www.ti.com
TI Distributors	focus.ti.com/docs/general/distributor.jhtml
Semiconductor Product Information Center (PIC)	support.ti.com
DSP Solutions	www.ti.com/dsps
DSP Knowledge Base Home Page	www-k.ext.ti.com/sc/technical-support/dsp-kbase
My TI homepage	www.ti.com/home_m_allmyti

To check for updated URLs, go to www.ti.com.

Americas

Product Information Center (PIC)	+1 (972) 644-5580
	Fax +1 (214) 480-7800
	support.ti.com/sc/pic/americas.htm
TI Literature Response Center U.S.A	+1 (800) 477-8924
U.S. Technical Training Organization	+1 (972) 644-5580
	www.ti.com/sc/docs/training/training.htm
U.S.A. Factory Repair/Hardware Upgrades	+1 (888)551-2110
Technical Support	support.ti.com
DSP Internet BBS via anonymous ftp	ftp://ftp.ti.com/pub/tms320bbs

6.8. Technical Training

Success today means meeting competitive standards of quality, variety, customization, convenience, and timeliness. No longer is productivity the single measure of success. At Texas Instruments, your success in the design process is foremost on our minds.

Technical advancement and stiff competition are constantly driving the need to shorten the cycle time from design concept to market availability. State-of-the-art skills and practical application techniques are required to implement ideas quickly with a high degree of quality and value.

To help you meet these challenges, TI offers the latest training and consultation on advanced technologies. This training includes lectures by skilled instructors and hands-on lab exercises using the latest TI development tools to accelerate your learning experience. Real-world examples show you how to apply TI's advanced technology to your system, helping you discover more productive ways of gaining that competitive edge.

For technical training information or registration:

- U.S.—call 1 (800) 336-5236, extension 3904, or (972) 917-3894.
- U.S. Technical Training Organization—www.ti.com/sc/docs/training/training.htm
- To reach the Europe Customer Training Fax Helpline— + 49 81 61 80 40 10

or visit us on the Internet to register for live or online training at: http://www.ti.com/sc/docs/training/training.htm

6.9. TI Product Information Center

The Texas Instruments Semiconductor Product Information Center (PIC) can assist you with technical and non-technical inquiries and services. A staff of professionals will listen to your needs, answer questions, order documentation, or direct you to the appropriate department.

You can contact the Product Information Center for North, South, and Central America,
located in Dallas, Texas at (972) 644-5580:

From 8:00 a.m. to 6:00 p.m., CST, Tuesday through Friday
From 9:30 a.m. to 6:00 p.m., CST, on Monday

You can also find the PIC on the Internet:

- **Home Page**—www-k.ext.ti.com/sc/technical-support/product-information-centers.htm
- **Americas**—www-k.ext.ti.com/sc/technical-support/pic/americas.htm
- **Europe, Middle East, and Africa**—www-k.ext.ti.com/sc/technical-support/pic/euro.htm
- **Asia**—www-k.ext.ti.com/sc/technical-support/pic/asia.htm
- **Japan**—http://www-k.ext.ti.com/sc/technical-support/pic/japan.htm

6.10. Texas Instruments Products and Application Solutions

These are TI Internet or alias hyperlinks you can use to obtain information on other Texas Instruments products and application solutions.

Products	**www.ti.com/subname**	**Applications**	**www.ti.com/subname**
Semiconductors:		Audio	www.ti.com/audio
Amplifiers and Linear	amplifier.ti.com	Automotive	www.ti.com/automotive
Clocks and Timers	www.ti.com/clocks	Broadband	www.ti.com/broadband
DSP - Digital Signal Processing	dsp.ti.com	Communications & Telecom	/home_a_communications_telecom
Data Converters	dataconverter.ti.com	Computers & Peripherals	/home_a_computer_peripherals
Interface	interface.ti.com	Consumer Electronics	/home_a_consumer_electronics
Logic	logic.ti.com	Industrial	www.ti.com/digitalcontrol
Microcontrollers (MCU)	microcontroller.ti.com	Medical	www.ti.com/medical
Power Management	power.ti.com	Security	/home_a_security

RFID Systems	www.ti-rfid.com	Space Avionics, & Defense	www.ti.com/military
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Video & Imaging	www.ti.com/video
Switches and Multiplexers	/home_p_switches	Wireless	www.ti.com/wireless
Temperature Sensors and Control ICs	/home_p_temp		
DLP® - TV, Projectors & Cinema		Display New Products	
Calculators & Educational Technology			

7. Legal Stuff

This legal material covers:

- Trademarks
- Code Composer
- TI Open Source Materials
- Eclipse Legal Links

Trademarks

Windows, Windows 98, Windows ME, Windows XP, Visual SourceSafe,Visual Basic, and Visual C++ are all trademarks and/or registered trademarks of Microsoft Corporation.

InstallShield and DemoShield are registered trademarks and service marks of InstallShield Software Corporation.

C2000, C5000, C54x, C55x, C6000, C6x, OMAP, TMS320, TMS320C54x, TMS470, DaVinci, Code Composer, DSP/BIOS, eXpressDSP, XDS560, and XDS510 are trademarks of Texas Instruments.

All other marks are the property of their respective owners.

Code Composer Legal

The software application programs included (the 'Licensed Programs') consist of these materials:

1. TI proprietary materials (the 'TI Proprietary Programs'), which are subject to the licensing terms set forth in the TI Open Source Materials (found in the folder, `X:\Program Files\Texas Instruments\CCSv4\docs\EULA\EULA.html`).

2. TI Open Source materials, which are subject to the licensing terms set forth in the Eclipse Public License version 1.0 (see TI Open Source Materials).

3. GNU materials, which are subject to the terms set forth in the GNU General Public License, Version 2.0 (GPL); included in the Code Composer EULA.

4. Apache materials, which are subject to the terms set forth in the in the Apache Software License, Version 1.0; included in the Code Composer EULA.

5. Java 2 Runtime materials, which are subject to the terms set forth in the in the Sun Microsystems, Inc., Binary Code License for the Java 2 Runtime Environment (J2RE) Standard Edition; included in the Code Composer EULA.

6. Mozilla materials, which are subject to the Mozilla Public License, version 1.1; included in the Code Composer EULA.

7. Logging Framework for C++ materials, which are subject to the terms of the Apache Software License, Version 1.0; included in the Code Composer EULA.

TI Open Source Materials

With the exceptions stated below, the TI Open Source Materials consist of (**i**) the files contained in the `\DebugServer\cc`, `\DebugServer\drivers`, `\DebugServer\examples`, `\DebugServer\scripting`, `\DebugServer\services`, and `\Tools\compiler` directories, as well as any associated subdirectories, if any, and (**ii**) all files, folders and associated subdirectories with file or directory names that begin with '`com.ti`' and that are located in the `\eclipse\plugins` or `\eclipse\features` folders.

These files *are not* TI Open Source Materials:
`\DebugServer\scripting\lib\org.eclipse.swt.win32.win32.x86_3.2.0.vTI.jar`, `\DebugServer\scripting\lib\swtwin323232.dll`, `\DebugServer\scripting\lib\rhino\js.jar`.

The TI Open Source Materials are licensed under the terms of the Eclipse Public License (EPL), Version 1.0, a copy of which is included with those materials. A copy of the EPL, ver. 1.0 is also available at http://www.eclipse.org/org/documents/epl-v10.php.

As a requirement of the EPL, TI hereby (**i**) disclaims on behalf of all `Eclipse.org` Contributors all warranties and conditions, express and implied, including warranties or conditions of title and noninfringement, and implied warranties or conditions of merchantability and fitness for a particular purpose; (**ii**) excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits; and (**iii**) states that any provisions which differ from the Eclipse Public License are offered by TI alone and not by any other party.

Eclipse Legal Links

- Eclipse Public License, version 1.0
- Eclipse Foundation Software User Agreement
- Trademark Attributions
- Guidelines for Eclipse Logos and Trademarks
- Legal Resources
- Committer Due Diligence Guidelines
- A Guide to the Legal Documentation for Eclipse-Based Content

Submit Code Composer Documentation Feedback

7.5. IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI’s standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265

Submit Code Composer Documentation Feedback