CCS Tips & Tricks

Updated for Code Composer Studio (CCS) version 10.4

Tips and Tricks

- General
  - Workspaces
  - Getting Started views
  - Windows and Views (basics)
  - Installing Eclipse Plug-ins
- Projects (Edit & Build)
  - Editor Tips
  - Indexer
  - Useful CCS Edit Views
- Debugging
  - Debug Configurations
  - Debugging Without a Project
  - Useful Debug Views

GENERAL

Eclipse Concept: Workspaces

- Main working folder for CCS
- Contains information to manage all the projects defined to it
  - The default location of any new projects created
- User preferences, custom perspectives, cached data for plug-ins, etc are all stored in the workspace
- Multiple workspaces can be maintained
  - Only one can be active within each CCS instance
  - The same workspace cannot be shared by multiple running instances of CCS
  - It is not recommended to share workspaces amongst users

Use Multiple Workspaces

- **Multiple Users**: Keep separate workspaces for each user on a shared machine
  - Custom preferences, layouts, etc will be maintained on a per user basis
  - Each user can be working on specific project(s) that would only be applicable to their workspace
- **Project Organization**: Break up all your CCS projects into separate workspaces for better maintenance
  - A workspace for each software release
  - A workspace for each module/feature of a release
- **Performance**: The larger the contents of the workspace (number of open projects), the greater the impact on performance of CCS

(Occasionally) Clean Your Workspace

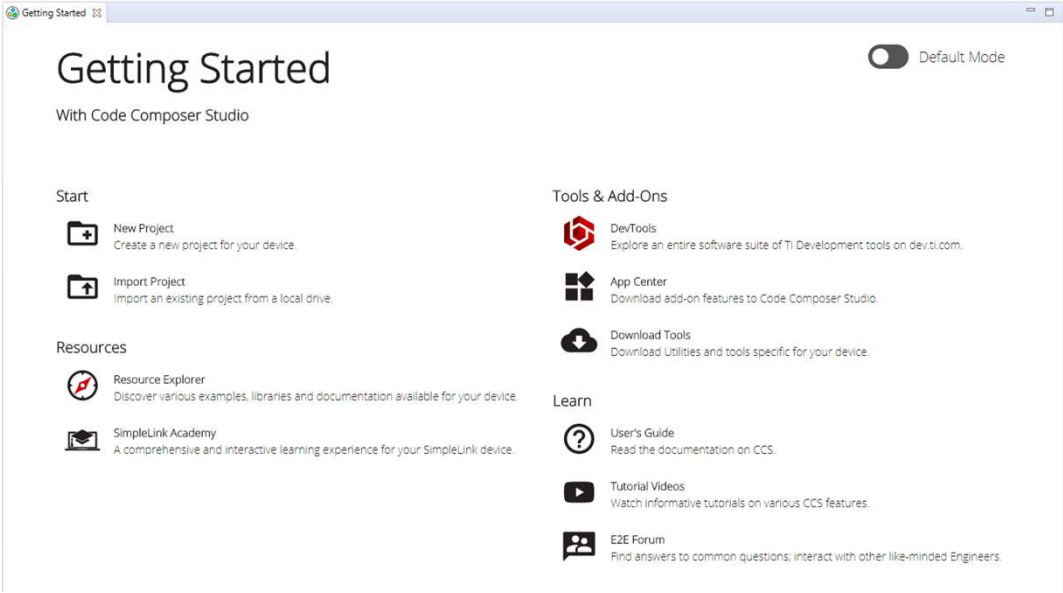
The workspace folder can get corrupted over time

Good idea to periodically clean your workspace for best CCS (Eclipse) performance and stability

- To clean workspace, either:
  - Delete the **.metadata** folder in workspace folder
  - Use a new workspace folder
- Before cleaning, save current workspace settings so they can be imported into the new workspace
  - Save settings: *File->Export...->General->Preferences->To preference file*
  - Import Settings: *File->Import...->General->Preferences->From preference file*
- Any projects will have to be re-imported after cleaning the workspace

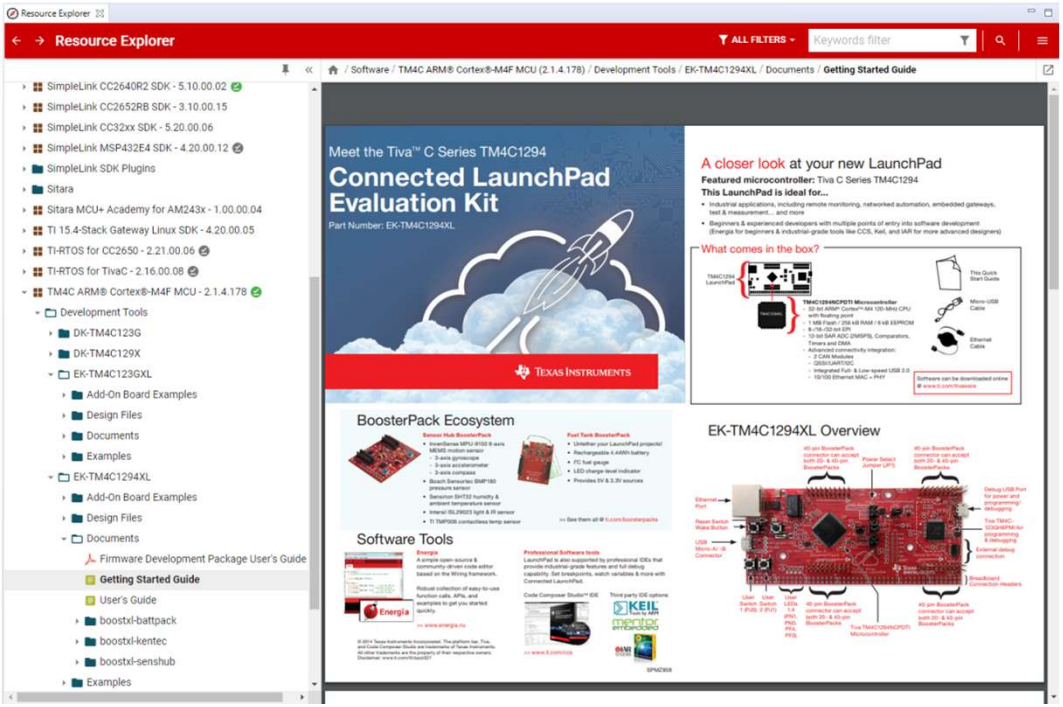
View: Getting Started

- Initial screen visible when Code Composer Studio is launched
- Provides fast access to common tasks
  - Create a new project
  - Use an example
  - Access the User's Guide…
- Links to support, videos and training material
- Available from the **View** menu



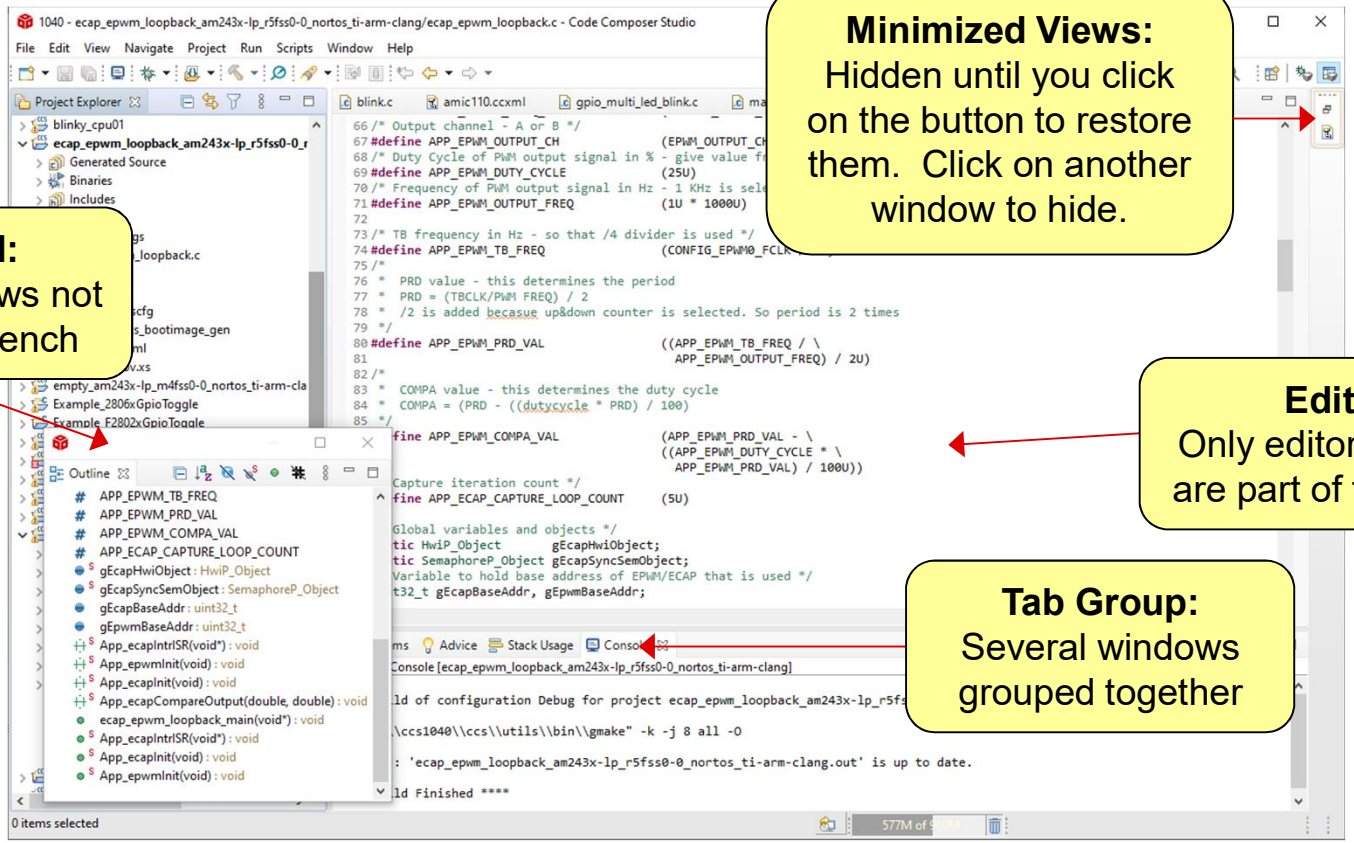
View: Resource Explorer

- Easily access a broad selection of packages such as C2000Ware, MSP430ware, SimpleLink SDK, Sitara SDK, TivaWare…
- View and import examples into CCS
- Provides links to documentation, videos and other collateral
- Accessible from both inside CCS and from any standard web browser via https://dev.ti.com/tirex



WINDOWS AND VIEWS

Window Types



Minimized Views:
Hidden until you click on the button to restore them. Click on another window to hide.

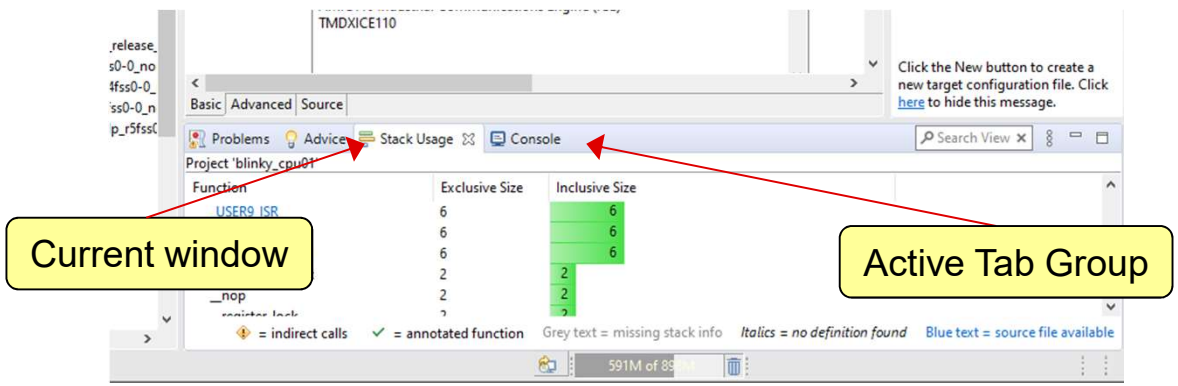
Detached:
Floating windows not tied to Workbench

Editor:
Only editor windows are part of this group

Tab Group:
Several windows grouped together

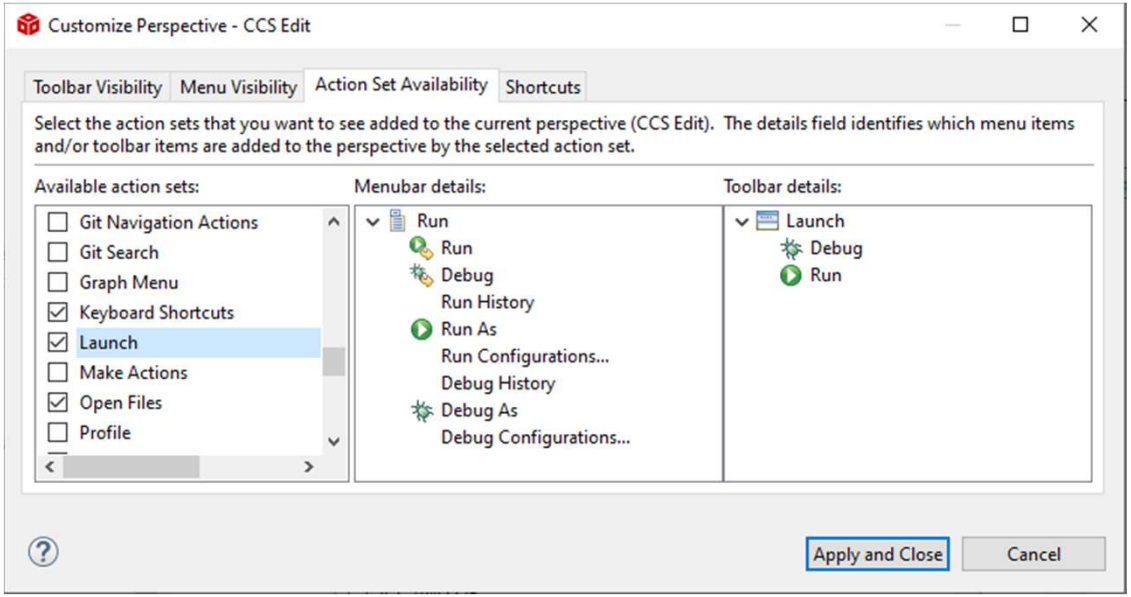
Windowing Tips

- Double-clicking on the title bar of a window will maximize the window
  - Double-clicking again will restore it to its previous size
- Minimized view stacks are great for windows you use infrequently but need a lot of space when you do use them
- The view that has focus is indicated by the active tab of the active Tab Group (indicated by the blue heading)



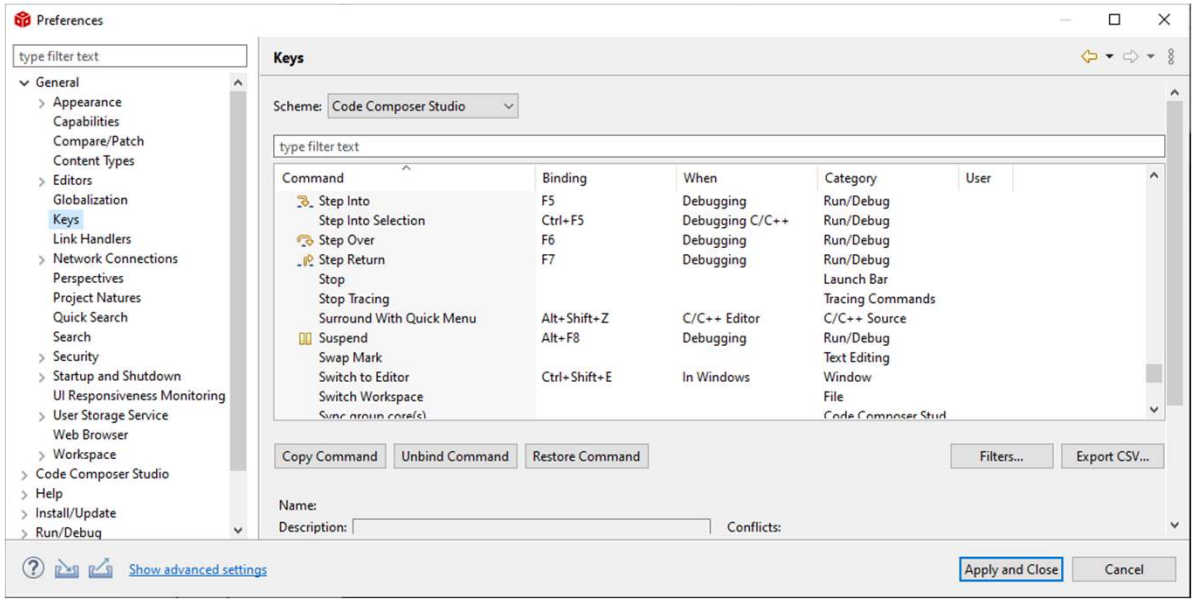
Customizing Perspectives

- You can customize the menu items and toolbars in your perspective
  - *Window -> Perspective -> Customize Perspective…*



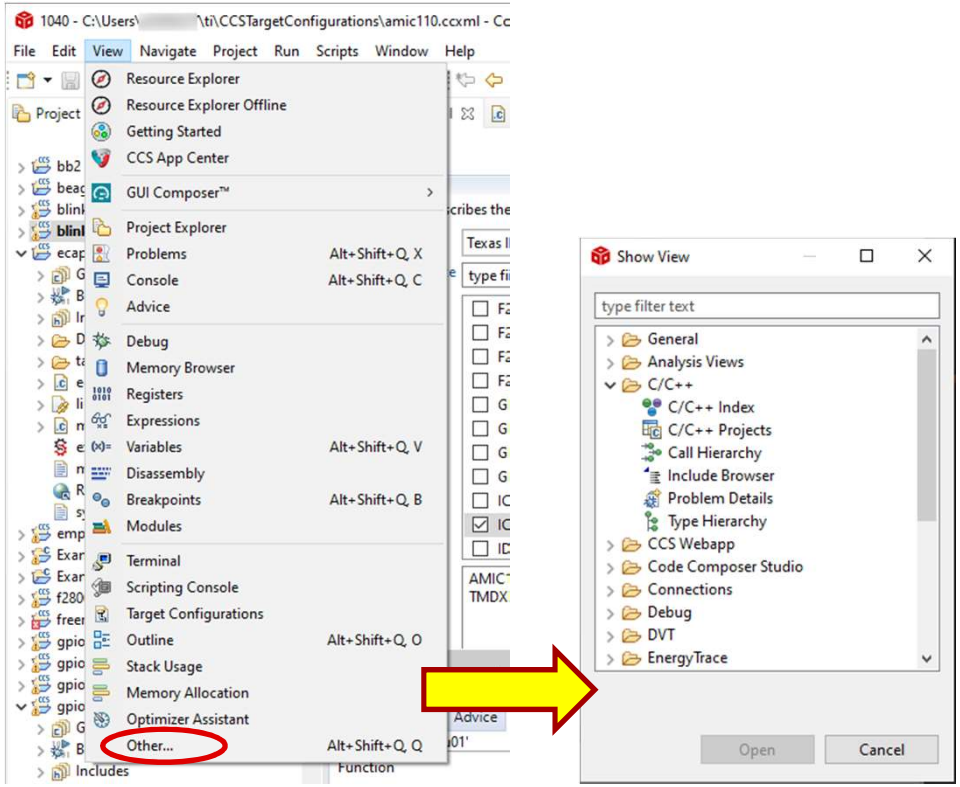
Using the Keyboard…

- All key bindings can be viewed and modified: *Window -> Preferences -> General -> Keys*
- Key bindings are part of the general preferences that can be exported to a preferences file and imported



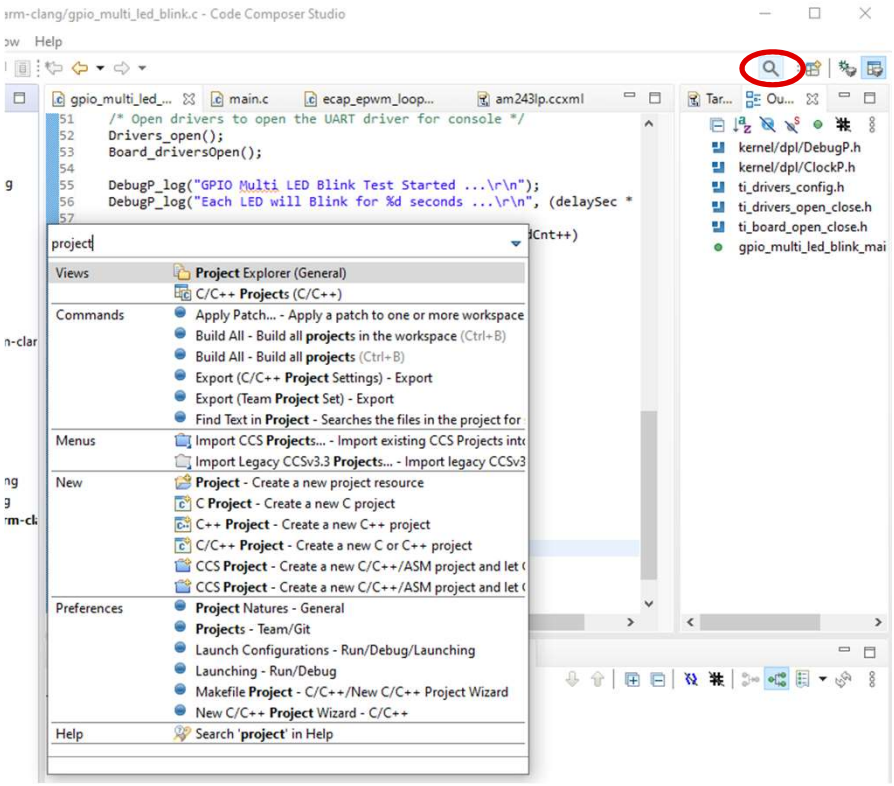
Accessing Views

- To open a new view go to the **View** menu
  - List the most commonly used views with CCS
- To access views that are not listed select **Other…**
- Many useful “hidden” views…



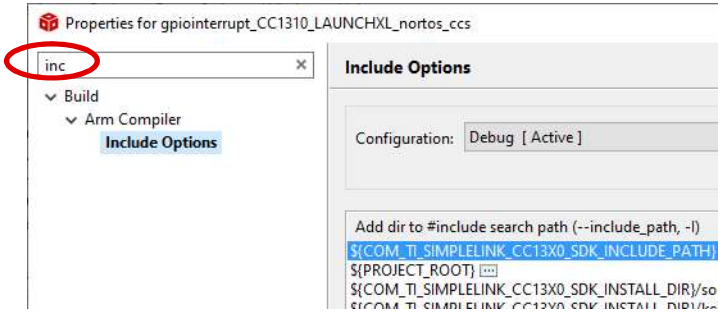
Find Actions

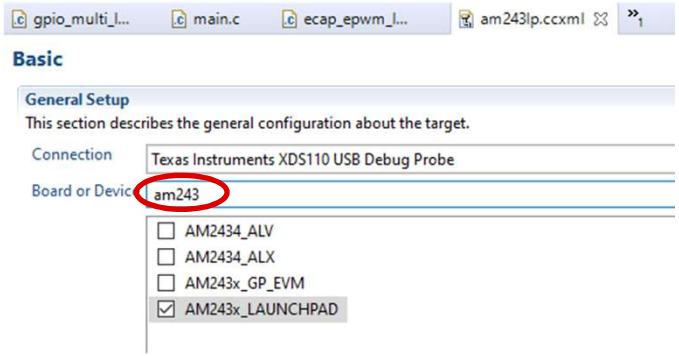
- Global search field that will search all of CCS
- Search for:
  - Views
  - Commands
  - Menu items
  - Preferences

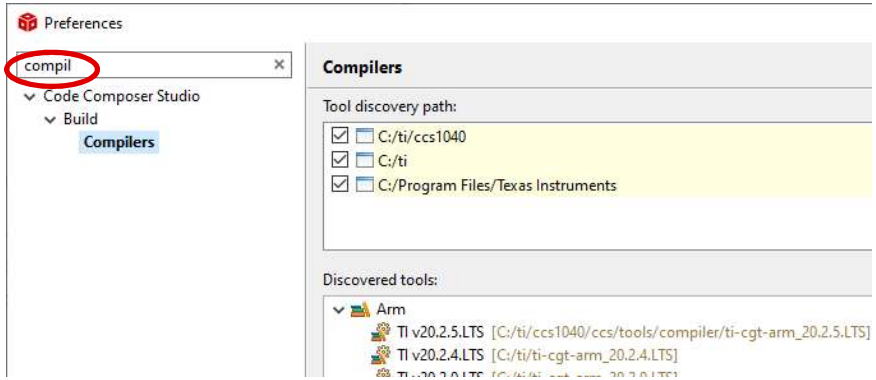


Filter Field

- Use it to find options/properties faster in the scope of a specific dialog/view
  - Narrows list of options down depending on characters entered in the field
- Available in many dialogs
  - Project Properties
  - Window Properties
  - Workspace Properties..
  - **Target Configuration** view







ECLIPSE PLUG-INS

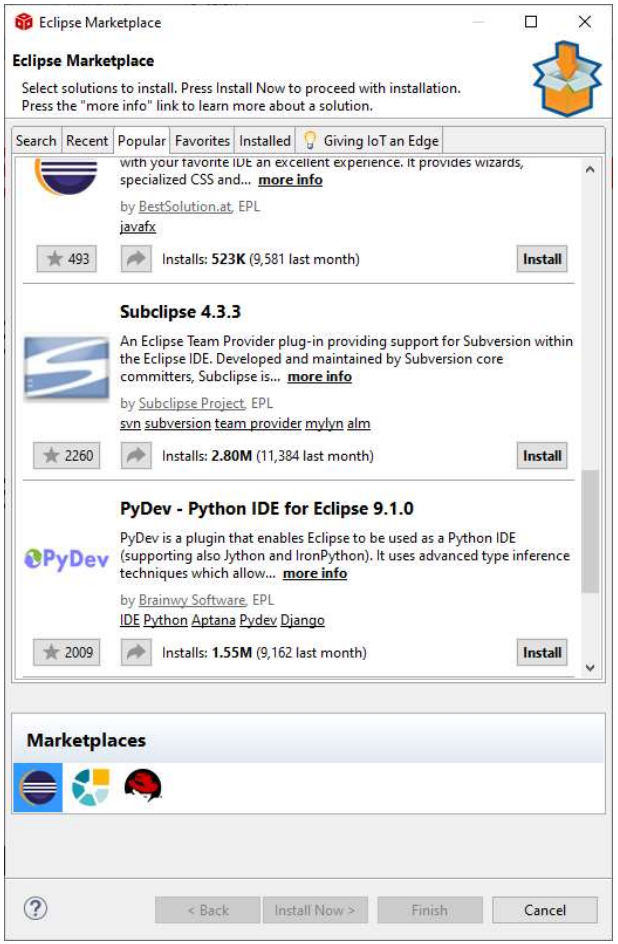
Eclipse Plug-ins - Basics

- CCS is based on Eclipse and is able to leverage many of the huge selection of 3rd party Eclipse plug-ins available
  - http://marketplace.eclipse.org/
- CCSv10.4 is based off Eclipse 4.14 and CDT 9.10
  - Look for plug-ins that support these versions for best compatibility
- CCS **App Center** focuses mostly on limited TI specific content and only shows a tiny fraction of available Eclipse plug-ins

⚠ Not all Eclipse plug-ins are compatible with CCS
  - Generic plug-ins that add functionality to the IDE have the best compatibility

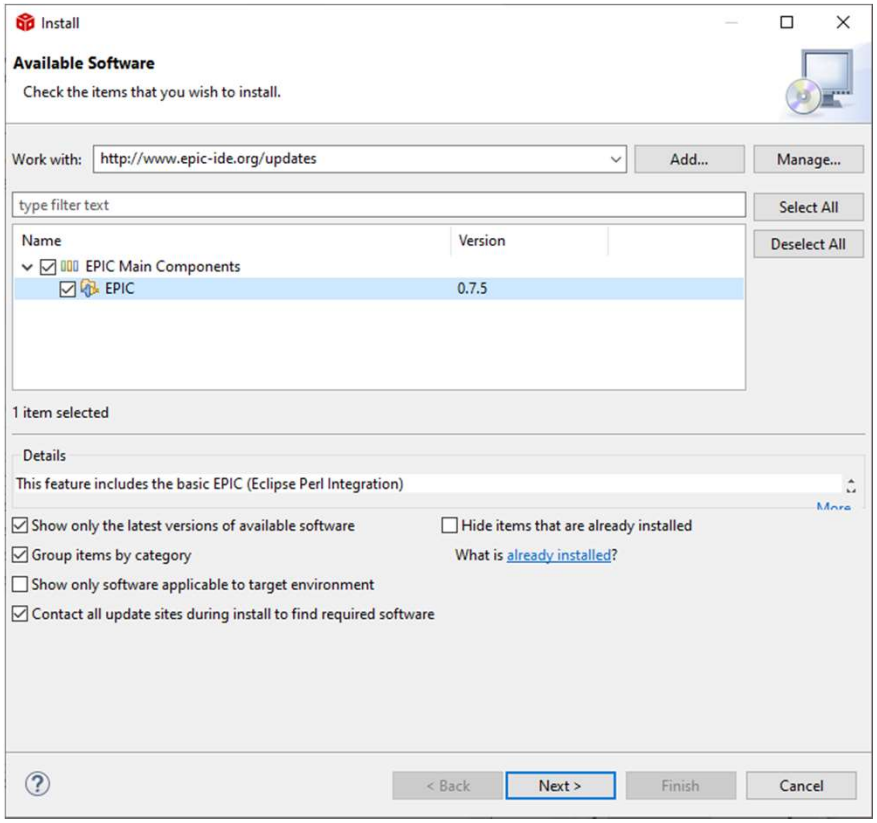
Eclipse Plug-ins - Marketplace

- Find additional Eclipse plug-ins from within CCS with the **Eclipse Marketplace** plug-in
  - *Help -> Eclipse Marketplace…*
  - Browse through a list of featured or popular plug-ins
  - Search for plug-ins with a keyword search
  - Easy install of any plug-in found in the marketplace



Eclipse Plug-ins - Installation

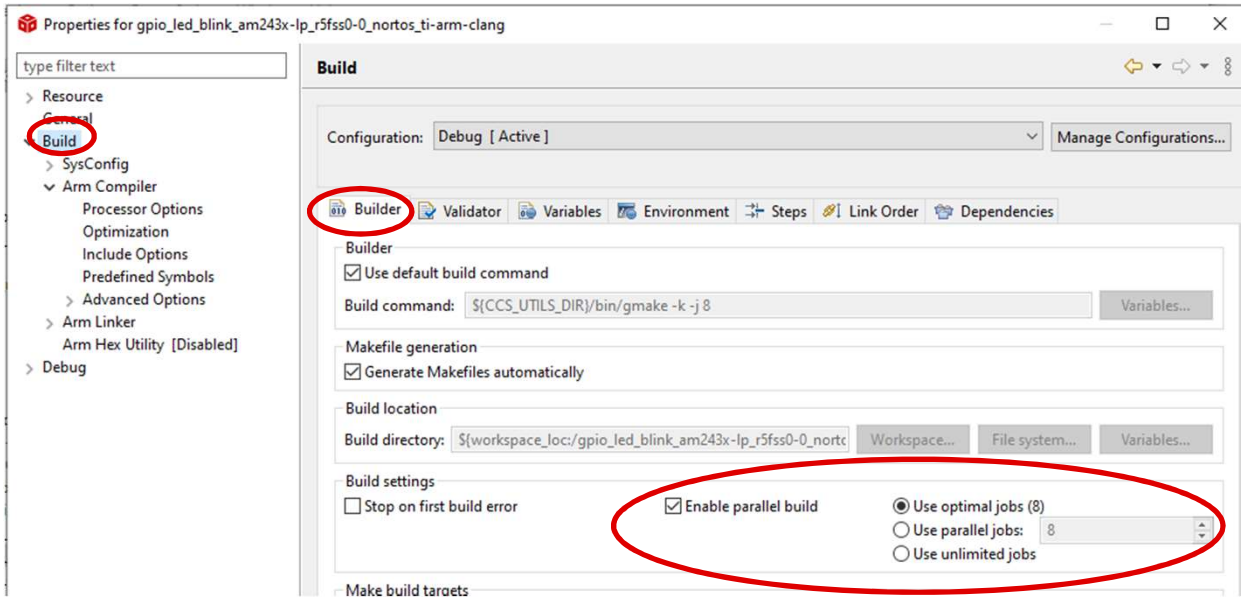
- Use the **Eclipse Update Manager** to install plug-ins if the plug-in update site is already known
  - *Help -> Install New Software* for new updates to install (specify remote site (URL) or local site (directory))
- Drop-in plugins manually
  - Many plug-ins can be simply downloaded as an archive and copied into the *.\ccs\eclipse\dropins* folder



PROJECTS
(EDIT & BUILD)

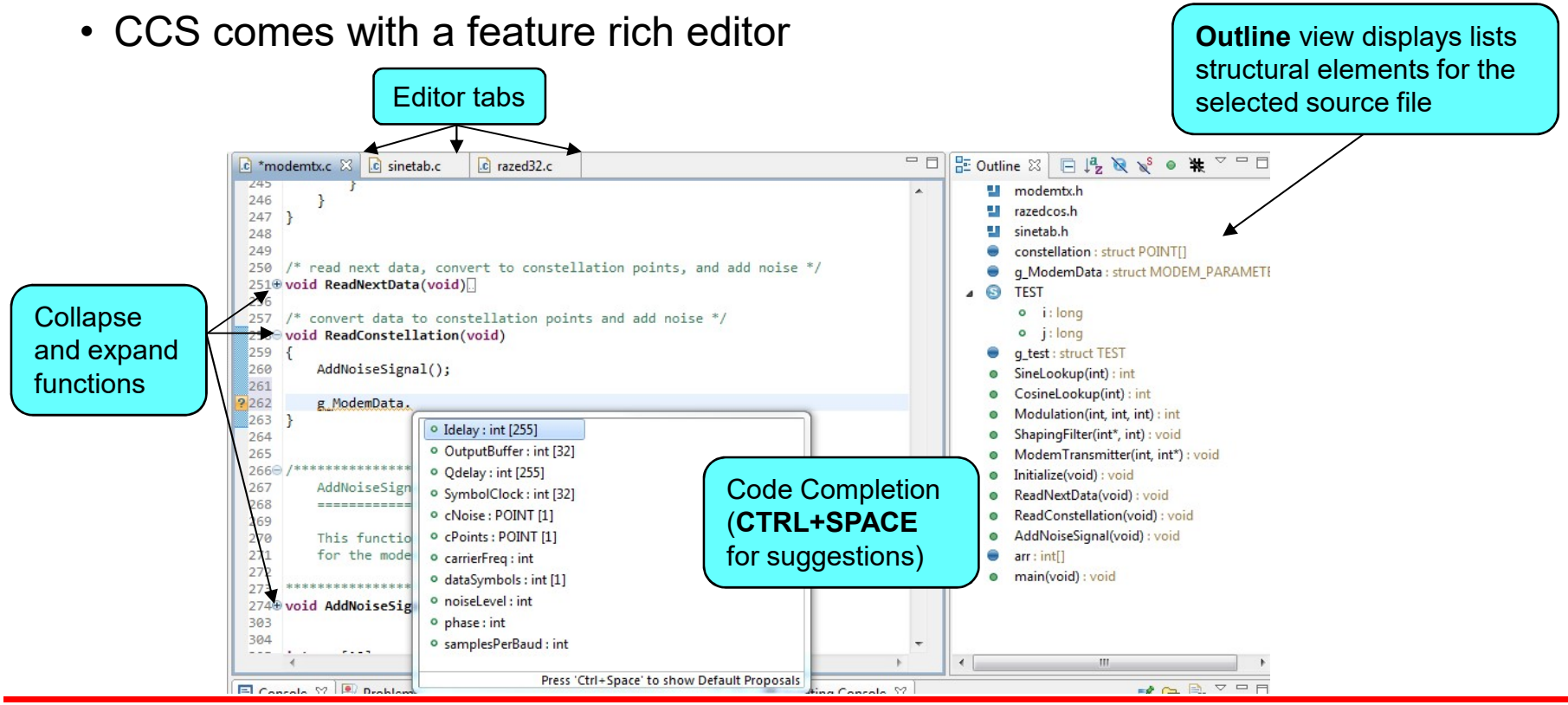
Parallel Builds

- Have a multi-core PC? Configure how many cores to allocate for project build
- In the project properties, under the **Builder** tab under **Build settings**, modify the parallel build options as desired (CCS will allocate all cores by default)



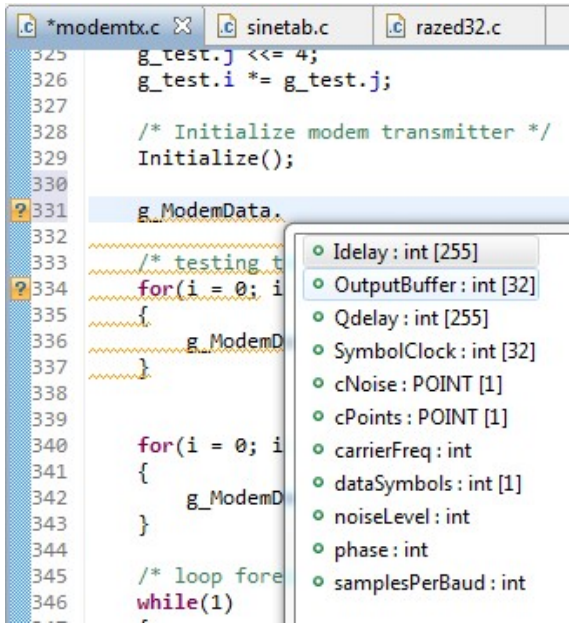
Editor: Source Code Editor

- CCS comes with a feature rich editor



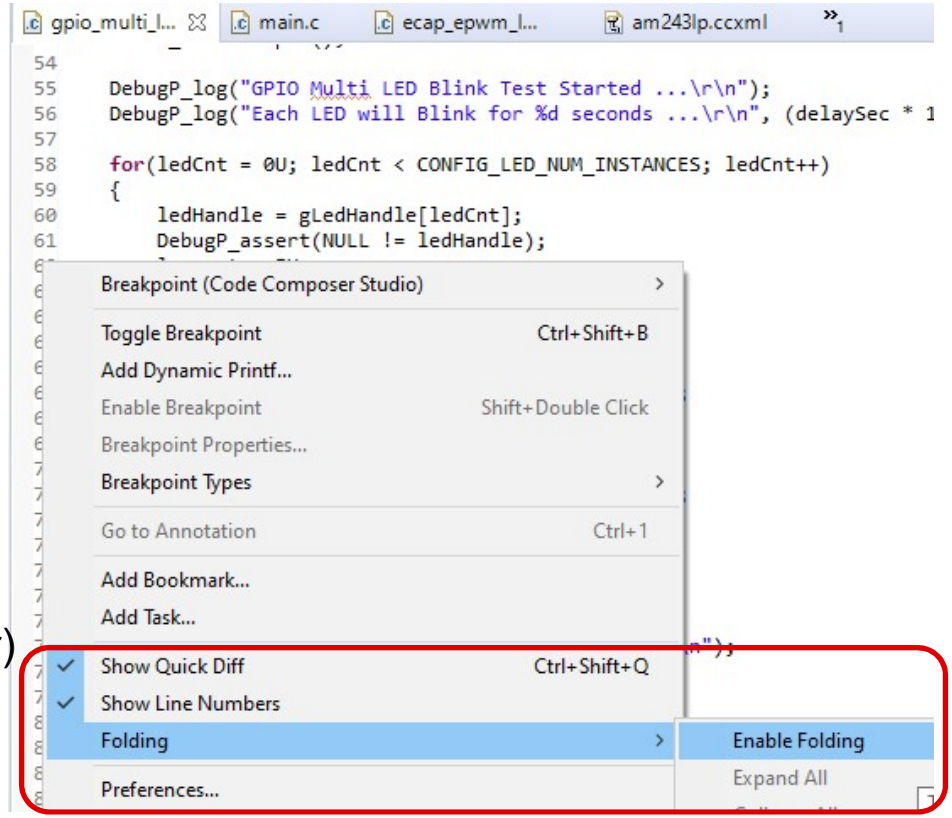
Editor: Advanced Editor Features

- Code Completion
  - Complete word
  - Auto-member information
  - Auto-parameter information
- Navigation
  - Back/Forward buttons
  - Back to last edit button
  - Go to definition
  - Go to declaration
- Code Folding
  - Collapse functions



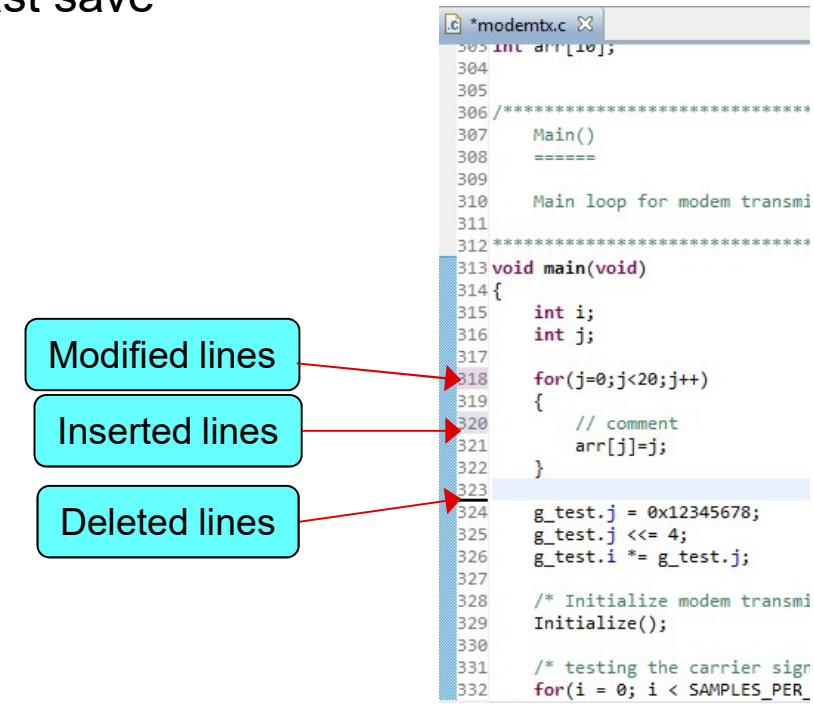
Editor: Advanced Editor Features

- Right-click in the editor margin to:
  - Toggle line numbers in the editor margin
  - Enable/Disable code **Folding**
  - Enable/Disable **Quick Diff**
  - Open editor **Preferences…** to access more options to configure:
    - **Content Assist** (Code Completion)
    - **Folding**
    - **Syntax Coloring**
    - **Hovers** (Cursor "hover over" behavior)
    - **Typing** behavior
    - etc



Editor: Edit Markers

- If you have the line number column on, it also indicates changes in your source file since your last save



Editor: Variable Highlighting

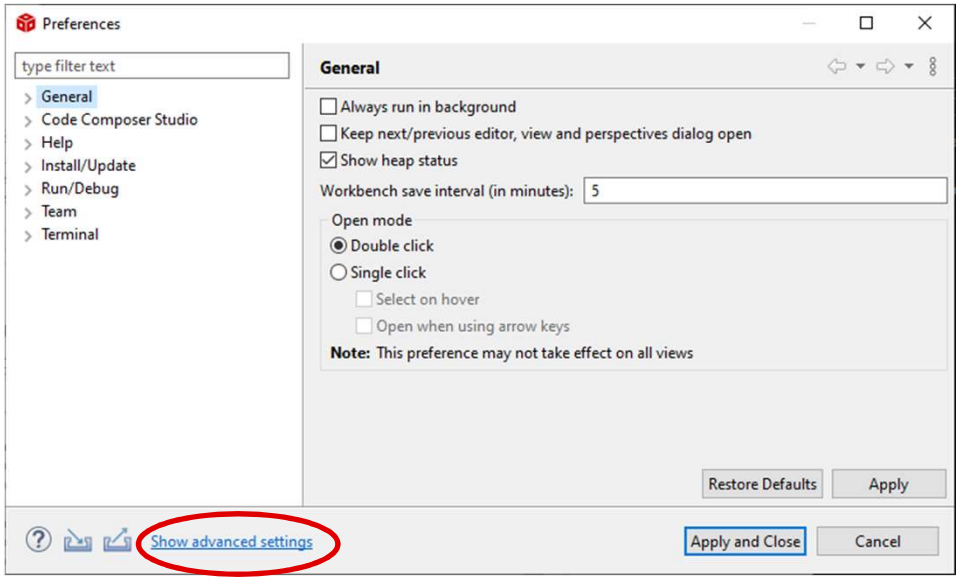
- Highlighting a variable in the editor will highlight all instances of the variable in the editor

modemtx.c

```
    }

    g_test.i = -16;
    g_test.j = 0x12345678;
    g_test.j <<= 4;
    g_test.i *= g_test.j;

    /* Initialize modem transmitter */
    Initialize();

    /* testing the carrier signal generator */
    for(i = 0; i < SAMPLES_PER_BAUD; i++ )
    {
        g_ModemData.OutputBuffer[i] = SineLookup(i*g_ModemData.carrierFreq);
    }


    for(i = 0; i < SAMPLES_PER_BAUD; i++ )
    {
        g_ModemData.OutputBuffer[i] = CosineLookup(i*g_ModemData.carrierFreq);
    }

    /* loop forever */
    while(1)
    {
        /* get modem data, convert to constellation points, and add noise. */
        ReadNextData();

        /* run modem on new data */
        for( i = 0; i < BAUD_PER_LOOP; i++ )
        {
            ModemTransmitter(i, &(g_ModemData.OutputBuffer[i*SAMPLES_PER_BAUD]));
        }
    }
}
```

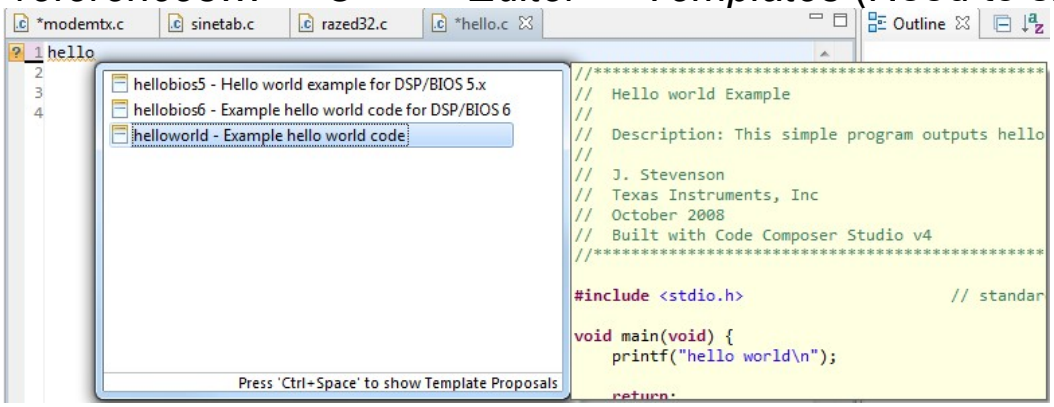
Preferences: Advanced Settings

- CCS hides some less commonly used “advanced” Eclipse/CDT settings
- Advanced settings can be enabled by selecting the **Show advanced settings** option in the lower left corner of both Workspace and Project level Preferences



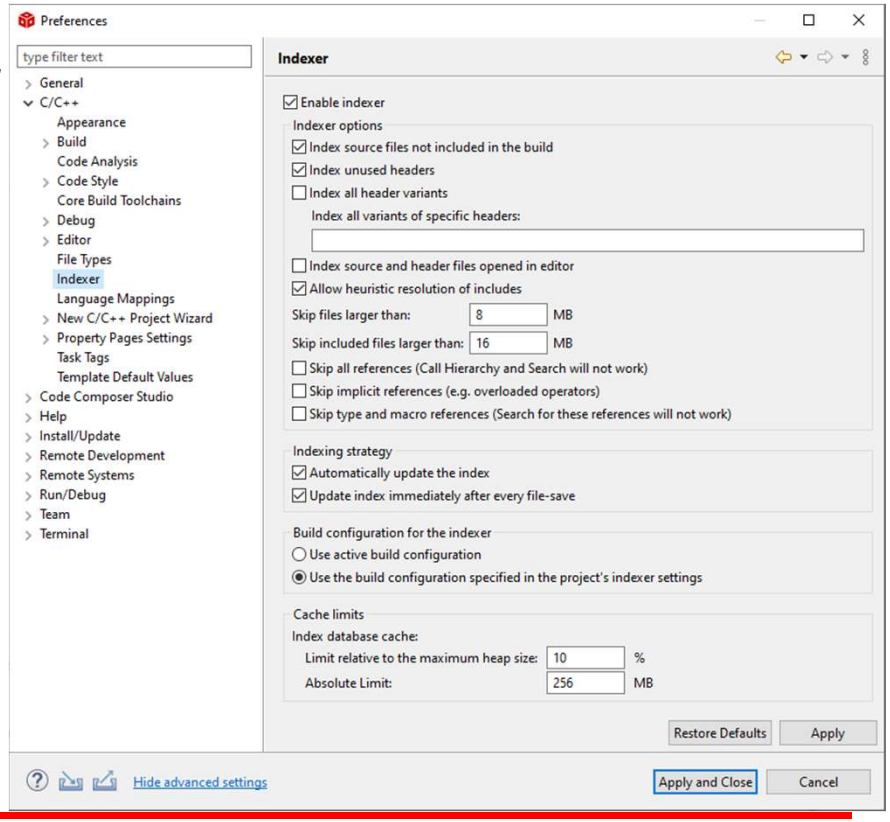
Editor: Source Templates

- CCS provides code templates
  - Ex: Hello World
    - Type in **h** in the editor and use **Content Assist** by pressing **CTRL+SPACE** keys (can also right-click in the editor and select **Content Assist** from the context menu)
  - Create custom templates for commonly used source code blocks or customize existing templates
    - *Window -> Preferences… -> C++ -> Editor -> Templates* (Need to show Advanced settings)



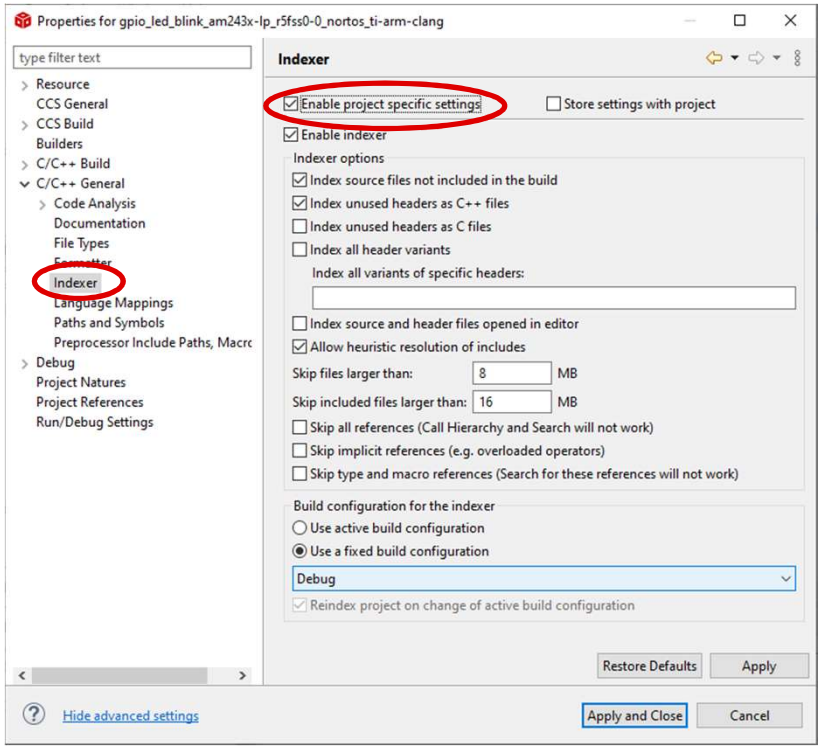
Indexer

- The advanced editor features rely on a database of the source and header files of the project that provides the basis for C/C++ search, navigation features and parts of content assist
(code completion)
- The C/C++ Indexer creates this database by parsing all of the source and header files of the projects open in the workspace
- Configure the Indexer:
  - *Window -> Preferences -> C/C++ -> Indexer* (Need to show Advanced settings)



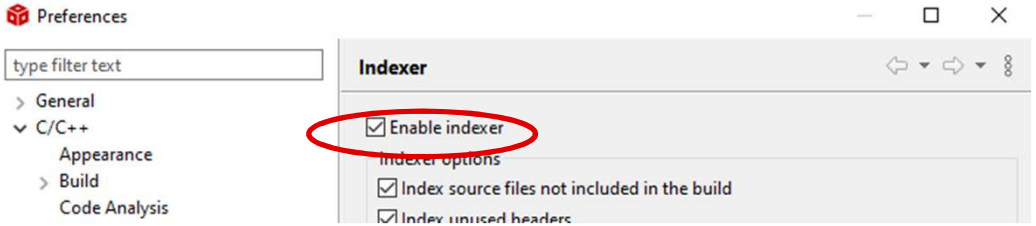
Indexer

- The Indexer can also be configured on a per project basis in the project properties
  - *C/C++ General -> Indexer*



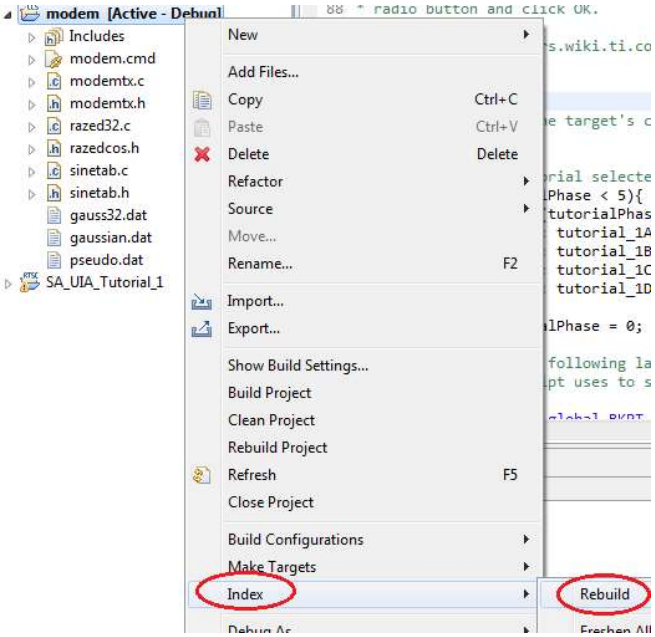
Performance Tip: Turn off the *Indexer*

- Don't use the CCS editor or don't need the advanced editor features?
- Turn off the Indexer!
  - The indexer constantly scans all open projects to support some advanced editor features
  - The indexer can use a decent amount of system resources, causing CCS to appear sluggish
    - This is most evident with large projects or many open projects in the workspace (or both)
  - The default setting is to have the indexer enabled



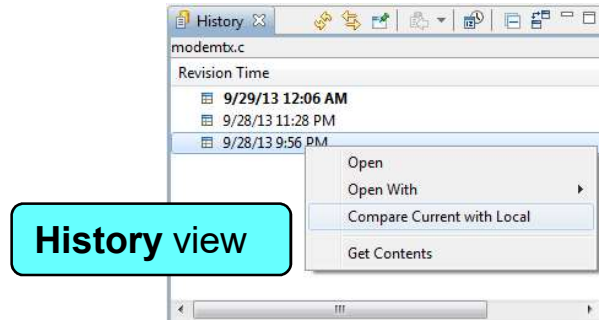
Troubleshooting Tip: Rebuild the *Index*

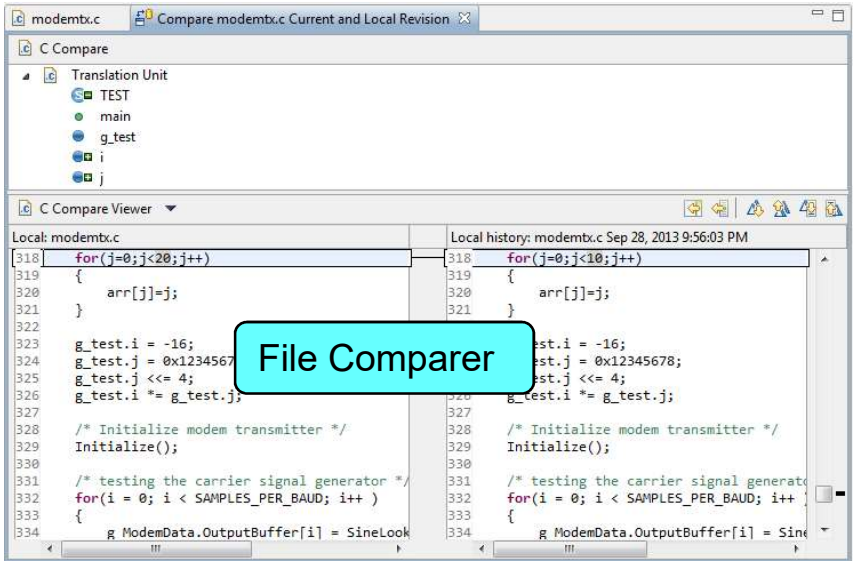
- Advanced editor feature not working right?
  - Code completion not working or bringing up the wrong suggestions?
  - Open declaration not finding the declaration?
  - Outline/Hierarchy views showing incorrect information?
- Indexed database/cache may have gotten corrupted or is out of date!
- Rebuild it!
  - Right-click on a project and select *Index -> Rebuild* to rebuild the indexed database for that project



View: History

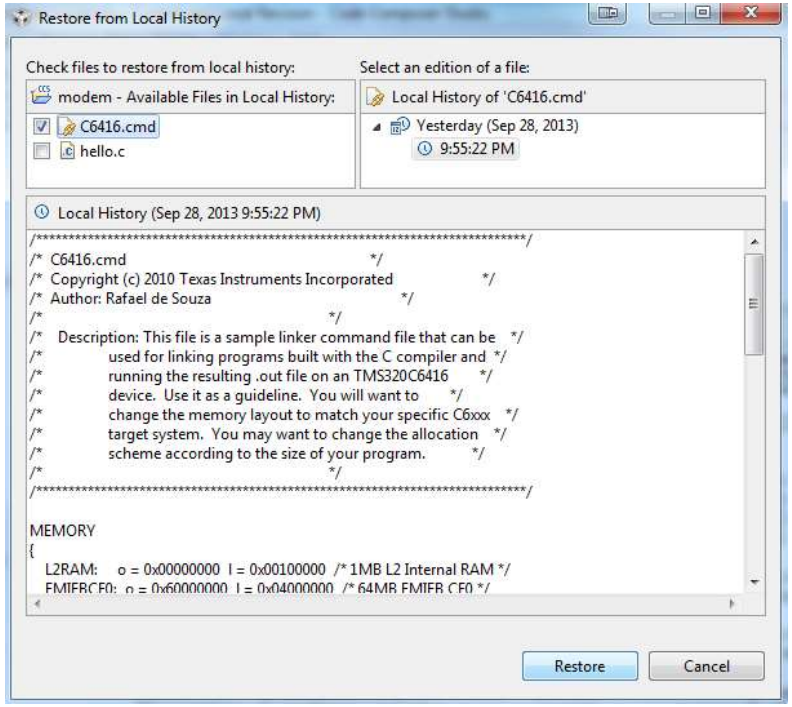
- CCS keeps a local history of source changes
  - Switch to the **CCS Edit** perspective
  - Right-click on a file in the editor an select *Team -> Show Local History*
    - Opens **History** view
- Use the **History** view to compare the current source file against any previous version or replace it with any previous version
  - Double-click on a revision to open it in the editor
  - Right-click on a revision to compare that revision to the current version





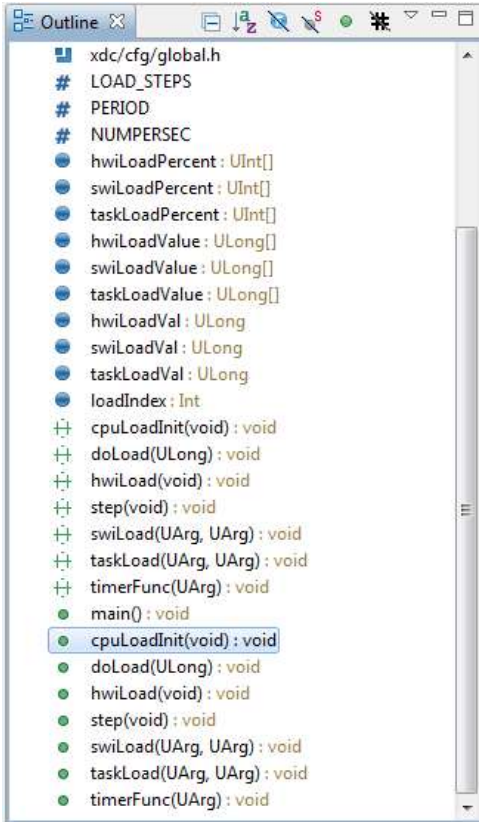
Local History - Project

- CCS keeps a local history of files for the project
  - Recover files deleted from the project
  - Right-click on the project and select **Recover from Local History** in the context menu



View: Outline

- Displays an outline of a structured file that is currently open in the editor area, and lists structural elements
- *View -> Outline*

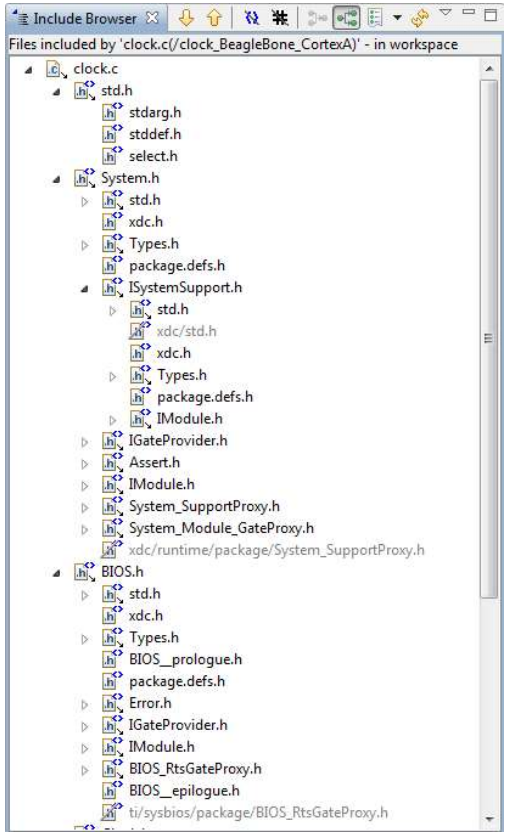


View: Include Browser

Which header files are this source file including?
Who is including this header file? Directly? Indirectly?

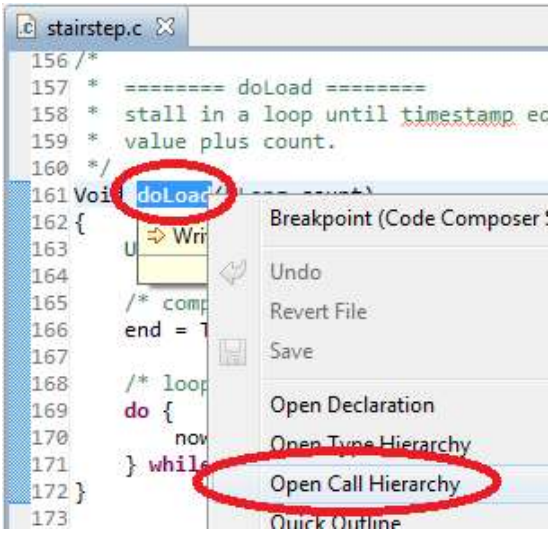
Show hierarchy of included header files for a source file

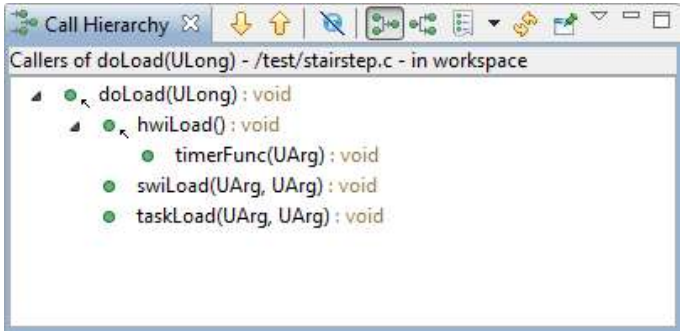
- *View -> Other… -> C/C++ -> Include Browser*



View: Call Hierarchy

- Displays callers and callees for a selected function
- Right-click on a function and then select **Open Call Hierarchy** in the context menu





View: Stack Usage

- Provides a static view of stack usage for your application
- *View -> Stack Usage*

Stack Usage

Search View

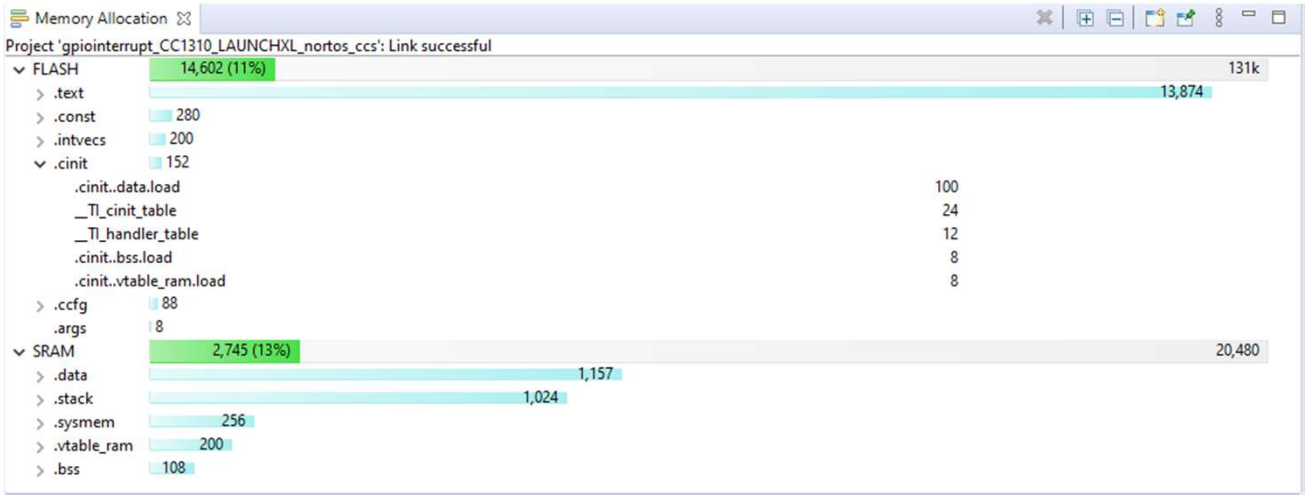
Project 'gpiointerrupt_CC1310_LAUNCHXL_nortos_ccs'

Function	Exclusive Size	Inclusive Size
gpioButtonFxn0	8	32
gpioButtonFxn1	8	32
PowerCC26XX_standbyPolicy	32	164
_c_int00	0	312
copy_in	*0*	
__TI_auto_init_nobinit_nopinit	16	16
_nop	0	
abort	0	
switchXOSCHFclockFunc	24	92
emptyClockFunc	0	
List_put	*0*	
List_remove	*0*	
PowerCC26XX_auxISR	40	140
PowerCC26XX_RCOSC_clockFunc	0	100

= indirect calls = annotated function Grey text = missing stack info *Italics = no definition found* Blue text = source file available

View: Memory Allocation

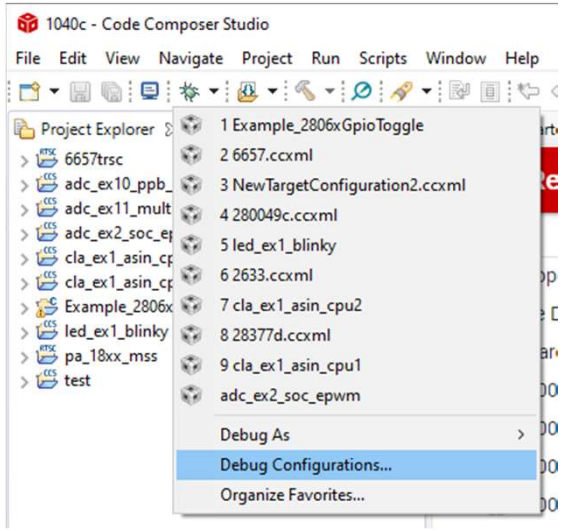
- Shows a graphical representation of how much memory is consumed by your application
- Expand each memory region to see how much memory each individual section or sub-section is using
- *View -> Other… -> C/C++ -> Include Browser*



DEBUGGING

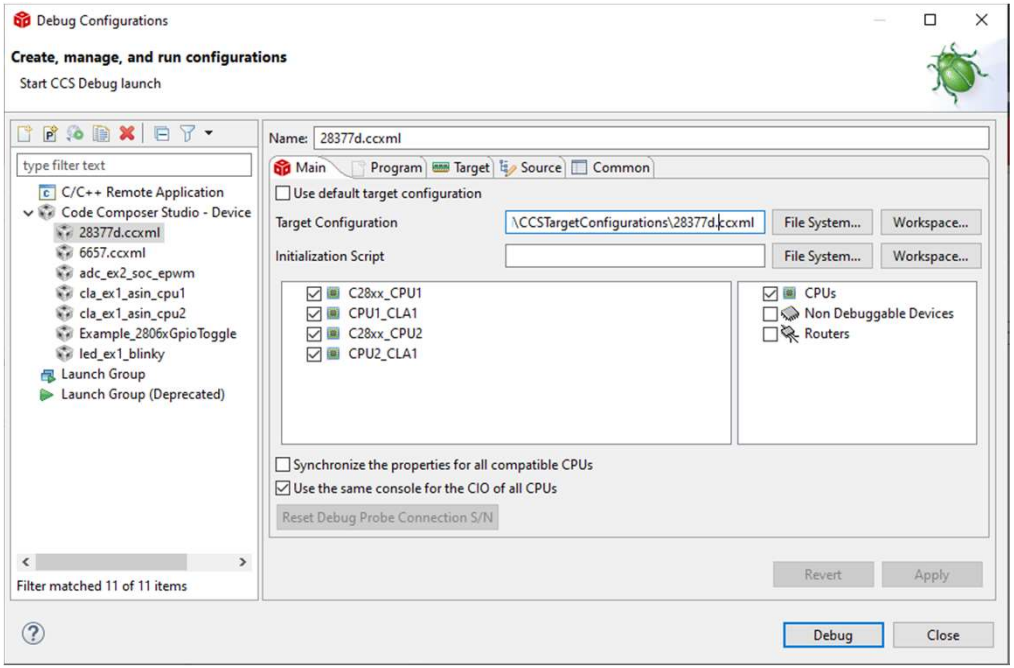
Debug Configurations

- Debug information is created when a debug session is first launched for a project or target configuration
  - Information stored includes which target configuration to use, debug settings…
- Both project and project-less debug sessions are launched using a debug configuration
  - A debug session can be started by explicitly launching a debug configuration
- To configure: Use drop-down menu next to the **Debug As** button and select **Debug Configurations…**



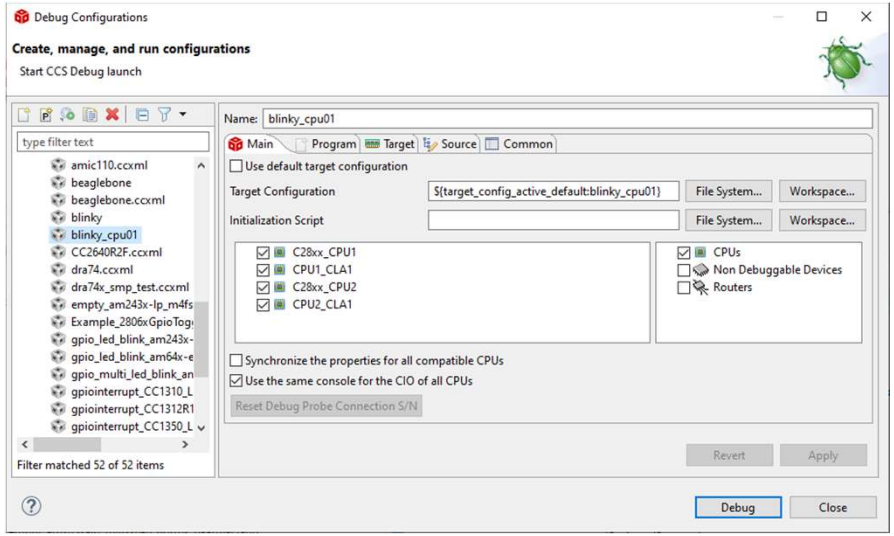
Debug Configurations

- Interface to manage existing configuration or to create new ones
- Existing debug configurations are configurable



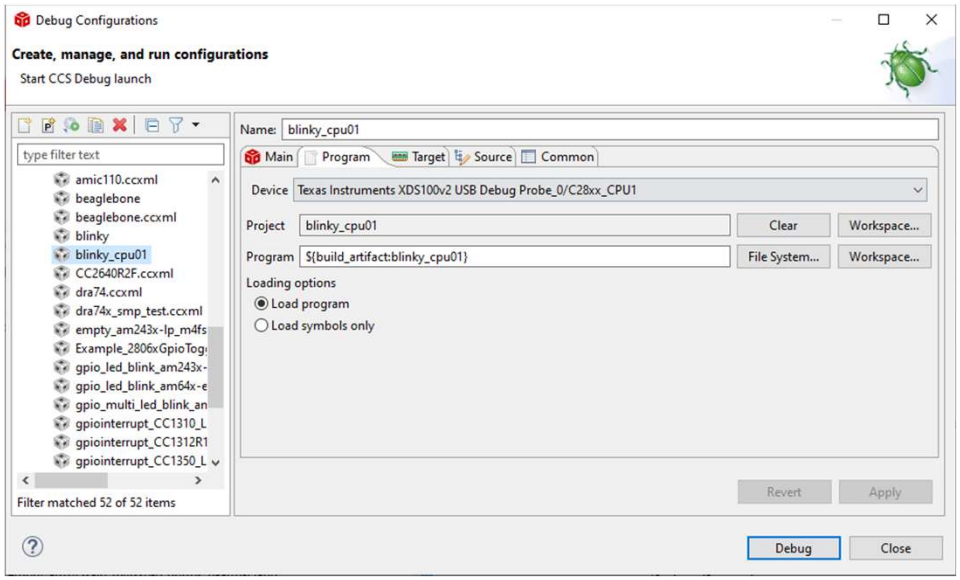
Debug Configurations: Main Options

- Use the **Main** tab to:
  - Which target configuration to use
  - Use the **Initialization Script** field to specify a DSS JavaScript for target initialization
  - Specify which devices on the JTAG scan chain will be visible by in the **Debug** view by default
  - Specify if all CPUs share the same console for C I/O (for multi-core debug)
    - C I/O will be interleaved in the same console (preceded with the CPU name)
    - Uncheck the option to create a separate C I/O console for each CPU



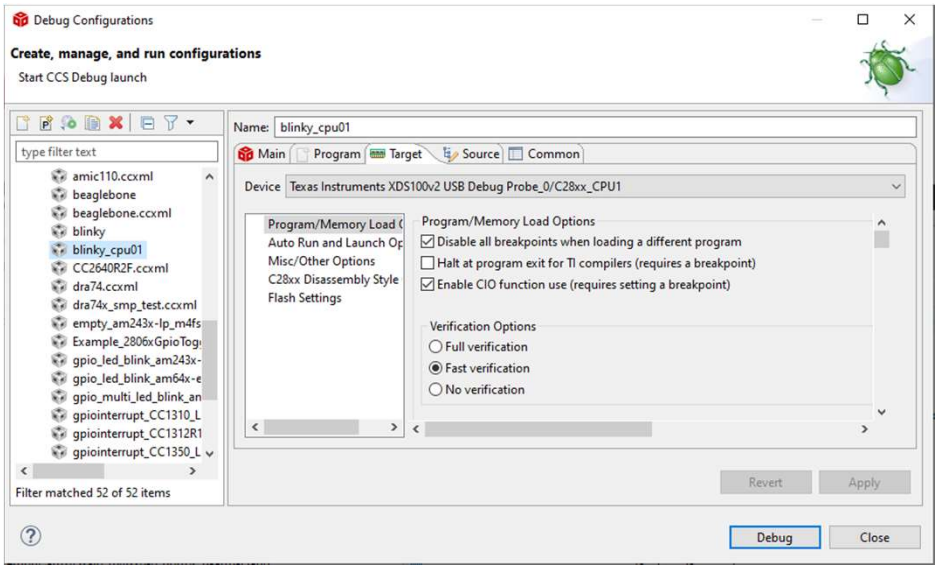
Debug Configurations: Program Options

- Use the **Program** tab to:
  - Specify which CPU to load the executable on (for multi-core devices)
    - Can specify different programs for each CPU
  - Specify to load the program (default) or just symbols only (to debug code in flash, etc)



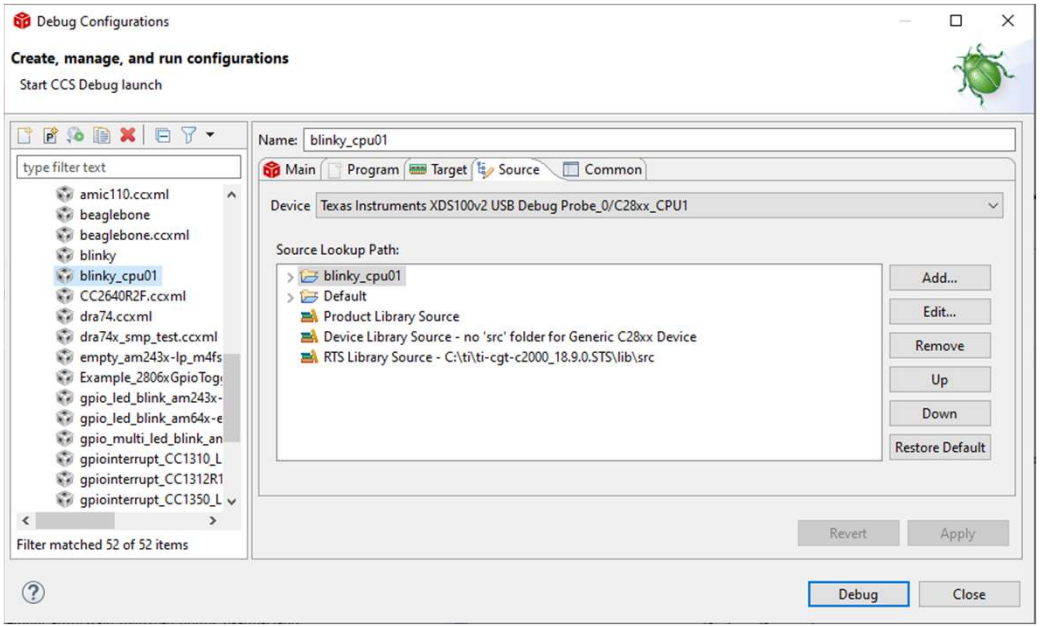
Debug Configurations: Target Options

- The **Target** tab can be used to set a variety of debug options like auto-run to *main*, auto-connect to a HW target, real-time options, program verification on load, etc…
- **Flash Programmer** options are available for applicable devices



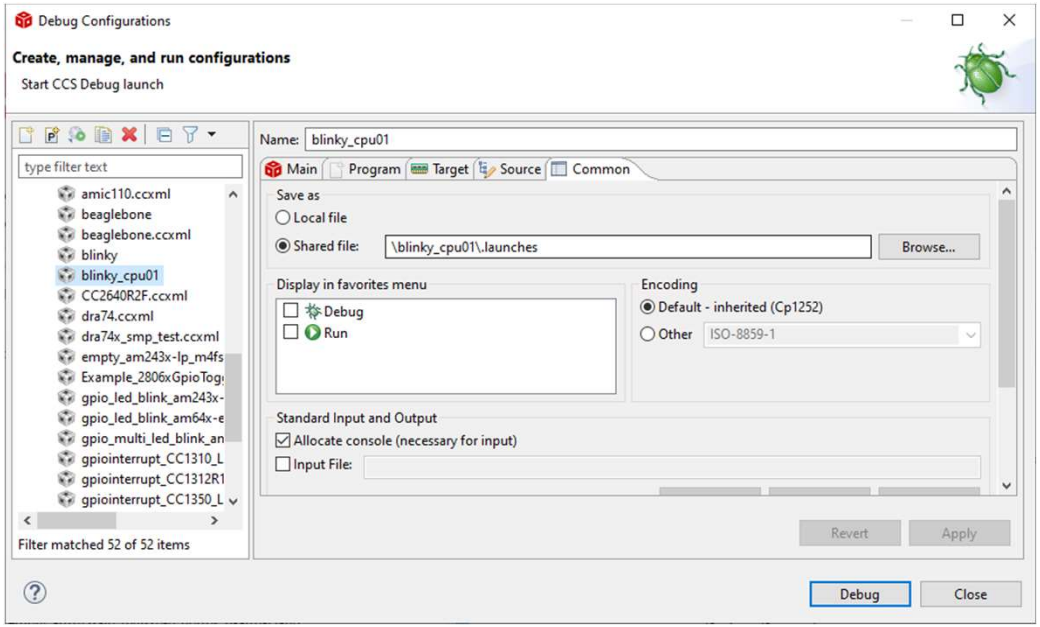
Debug Configurations: Source Options

- The **Source** tab allows you to add additional source lookup search paths
  - Specify paths on a per CPU granularity (for multi-core devices)
  - All paths to any source files in your project are automatically added by default



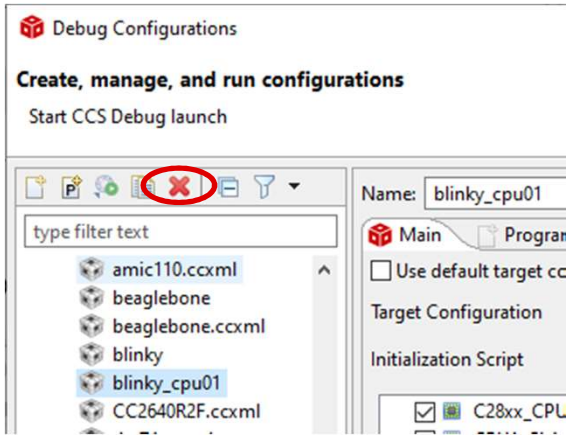
Debug Configurations: Common Options

- The **Common** tab contains a collection of miscellaneous options
  - Can specify the debugger to send all CIO to a file instead of the console
  - Specify character encoding



Troubleshooting Tip: Debug Configuration

- Having some strange debugger issues?
  - Having problems launching a debug session?
  - Can’t connect to the target anymore?
  - Is the debug session unstable?
- Debug Configuration may have gotten corrupted!
- Delete it and have CCS generate a new one
  - Select the **Debug Configuration** to delete in the **Debug Configurations** dialog and press the delete button



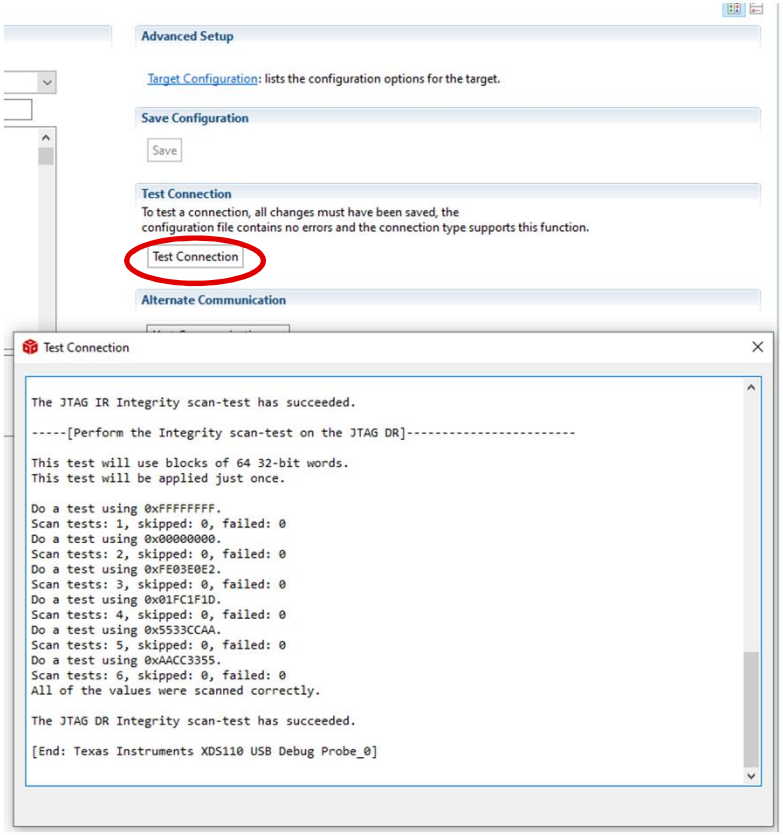
Troubleshooting Tip: Test Connectivity

Having target connection issues?

Validate your physical JTAG connection to the device from the driver

- Use the **Test Connection** button in the **Advanced** tab of the target configuration
  - Will test both the target configuration file and the JTAG connectivity between CCS and the target by running some diagnostics
  - If all test pass, then the physical connection between the device and the driver is fine and the issue is with the debugger
  - If there are failures, send the results to TI support

Not all debug probes support this feature

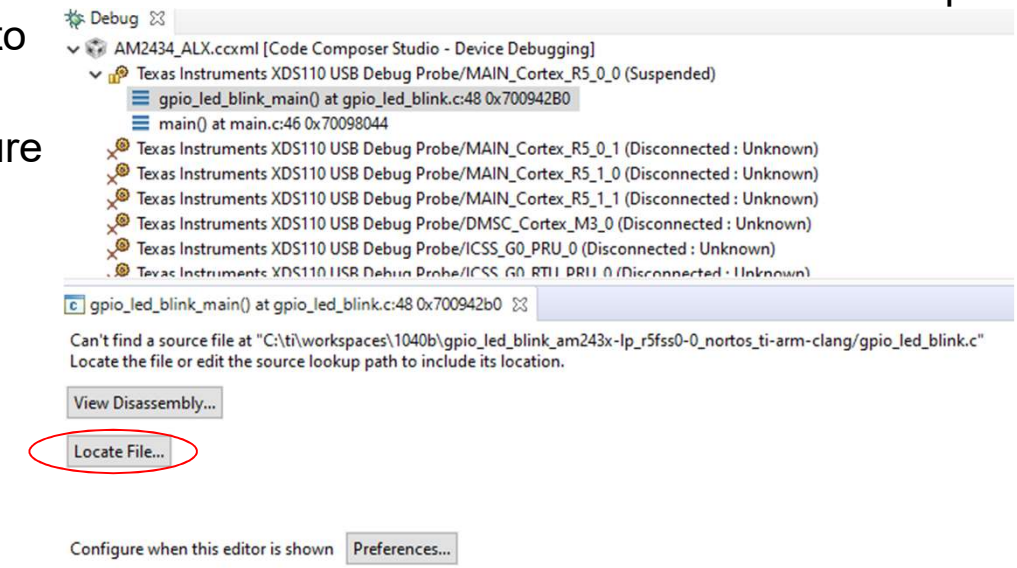


Debugging Without a Project: Source Lookup

- For project-less debug sessions, CCS will look for source files using relative path information stored in the debug symbols
  - CCS will find the source files if the executable and source files are in the exact same location as when the executable was originally built
- If the location of the executable file or source (or both) has changed, CCS may not be able to find the source files
- CCS can be instructed where to find the source files one of two ways:
  - Tell CCS where the first file is and let CCS find the rest of the files using relative path information in the symbols (recommended method)
  - Set **Source Lookup Paths** for CCS to scan when looking for source files:
    - Set for **Debug Configuration** - apply for every debug session launched by the debug configuration (under the **Source** options)
    - Set at global (workspace) level – apply for any debug session started with this workspace

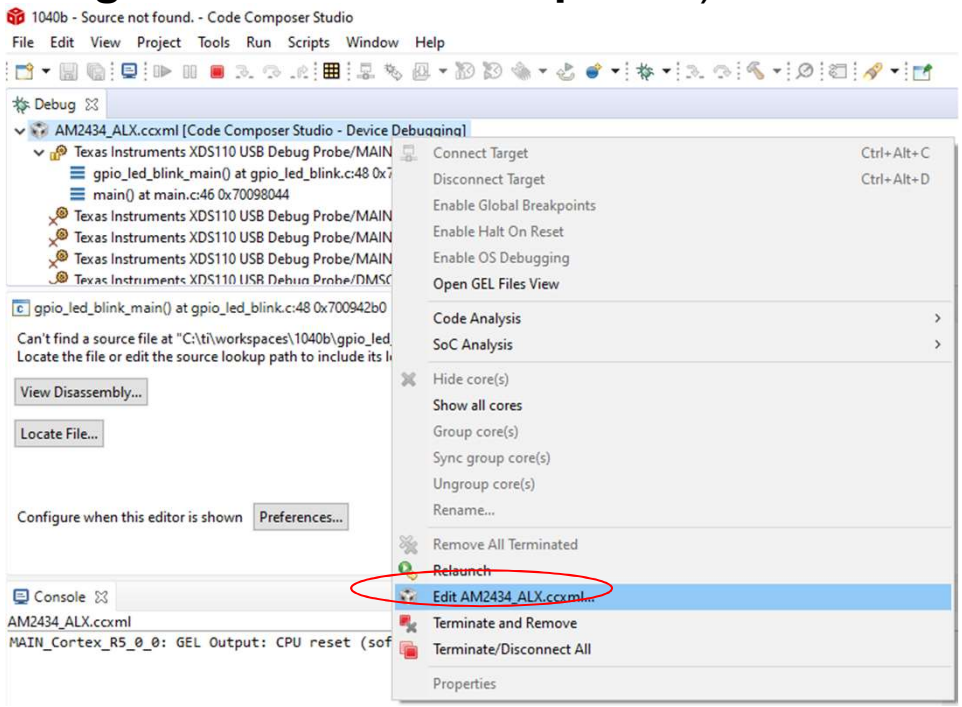
Source Lookup: Method #1 - Locate File…

- If a source file cannot be found during debug, it will be indicated in the editor
- Use **Locate File…** button to browse to the location of the source file
  - The debugger can then find other source files in the same location or use relative path information to find files relative to the current file
  - Location is remembered for future loads of the same program
- This method has the best performance for finding source files, assuming the original directory structure is intact



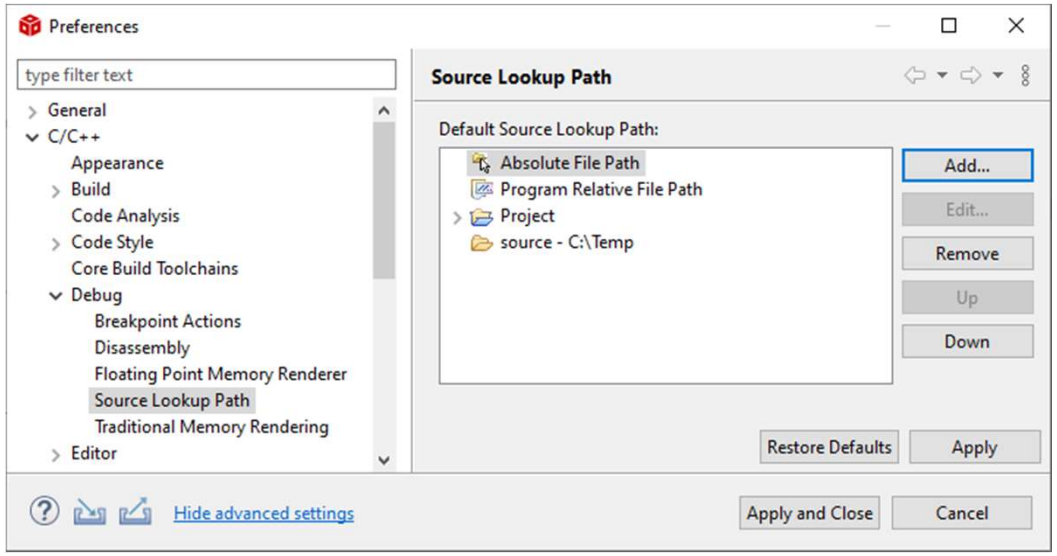
Source Lookup: Method #2 - Debug Configuration

- The **Source** tab in the debug configuration allows you to add additional source lookup search paths (see **Debug Configurations: Source Options**)
  - All paths to any source files in your project are automatically added by default
- **Debug Configurations** may also be accessed during an active debug session by right-clicking in the **Debug** view and selecting **Edit <Debug Configuration>** in the context menu



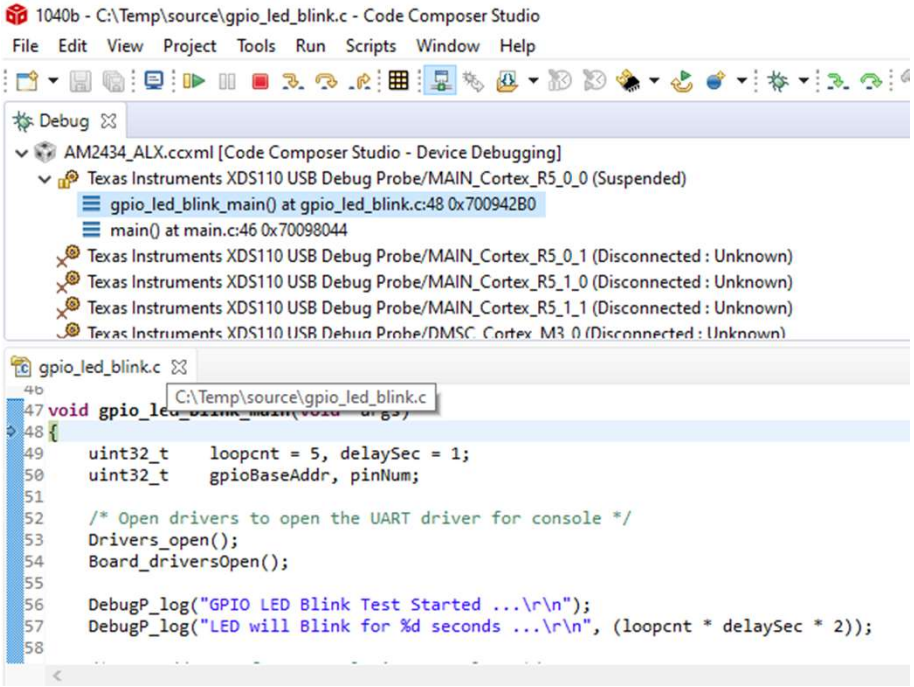
Source Lookup: Method #3 - Global (Workspace)

- Source lookup paths can also be set globally to apply for all debug contexts (in a multi-core environment) and debug sessions
  - *Windows -> Preferences -> C/C++ -> Debug -> Source Lookup Path* (Need to show Advanced settings)



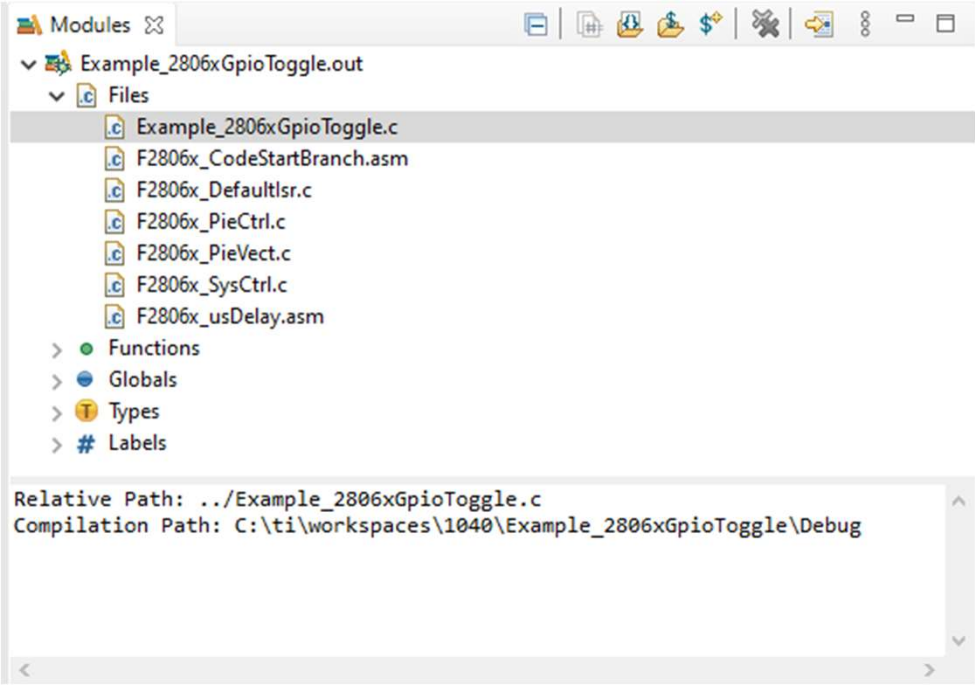
Source Lookup Paths

- Once the path is known to the debugger (using any method), the source file will be opened in the debugger
- Method #1 is recommended due to having the best performance
- The other methods may do recursive searches inside the specified directories when searching for files. If the directories have many subfolders and many files inside, the search may be slow and thus lead to slow performance when looking for the source files
- All previously mentioned methods can also be used to find additional files for project debug session (debugging code from a library)



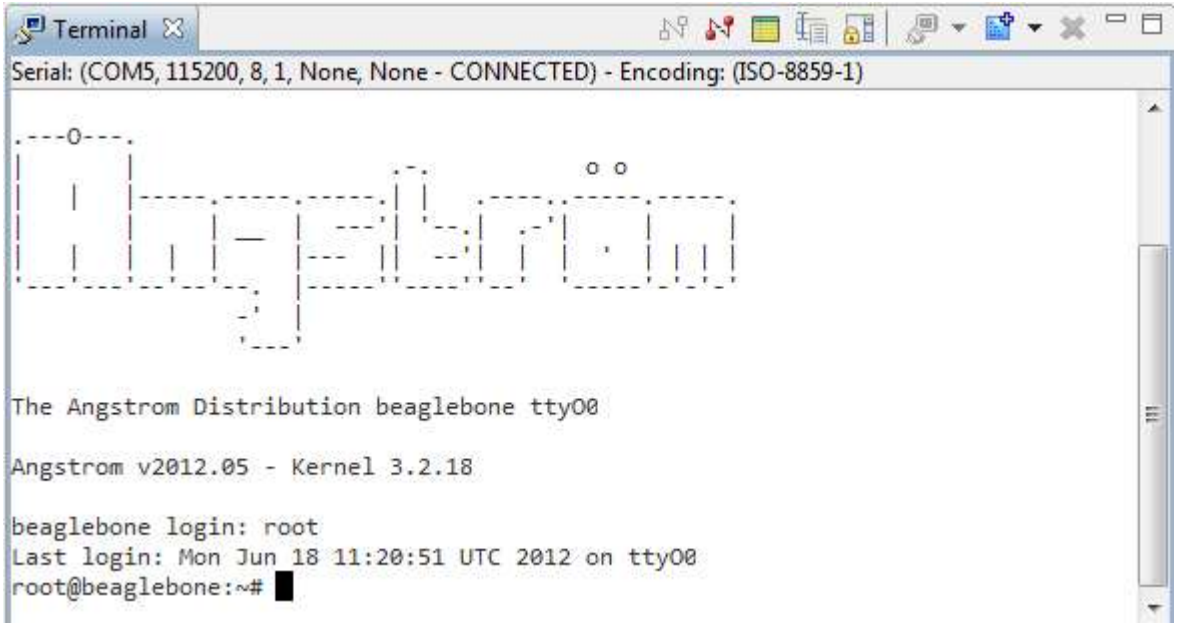
View: Modules

- Provides information for all loaded symbol files
- *View -> Modules*



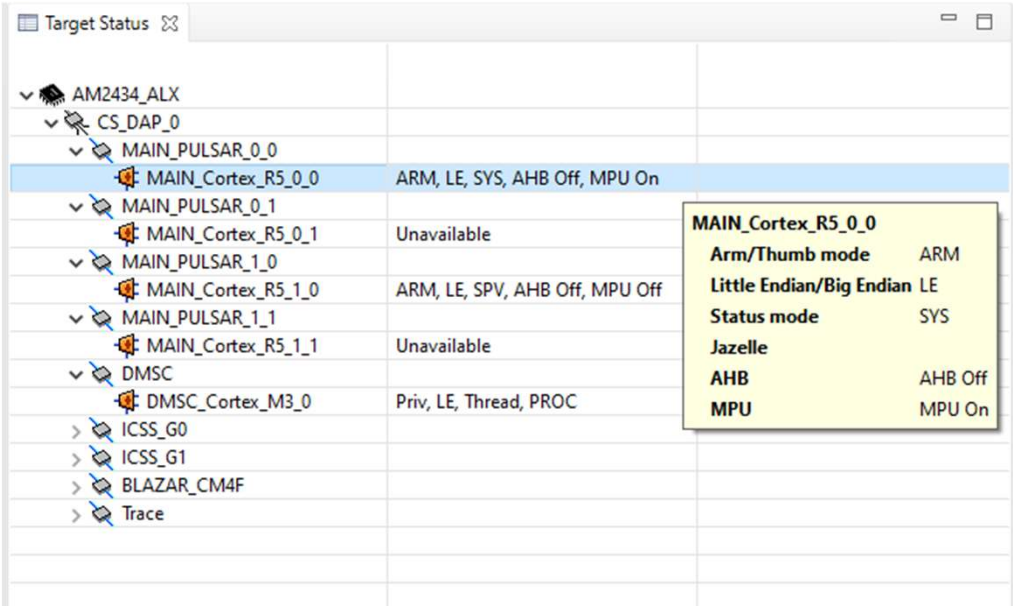
View: Terminal

- Terminal emulator that can connect to a remote target via a serial port or over TCP/IP using the TELNET or SSH protocol.
- *View -> Terminal*



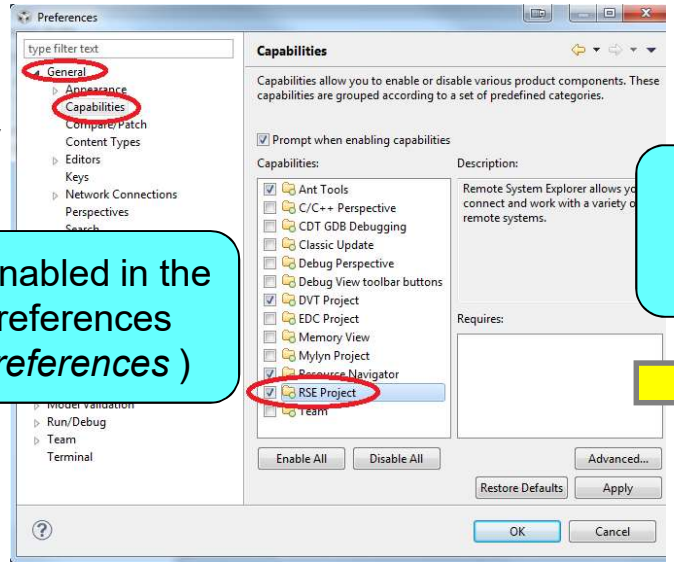
View: Target Status

- High level view of the status of all cores and non- debuggable devices of the system
- *View -> Other… -> Debug -> Target Status*

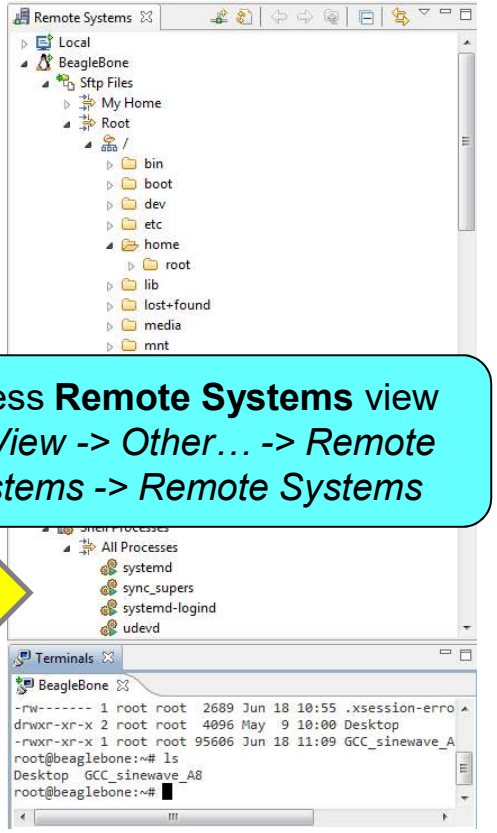


View: Remote Systems

- Remote Systems Explorer (RSE) is an Eclipse plug-in that provides:
  - Drag-and-drop access to the remote file system
  - Remote shell execution
  - Remote terminal
  - Remote process monitor



RSE must be enabled in the workspace preferences (*Window -> Preferences*)



Access **Remote Systems** view via *View -> Other… -> Remote Systems -> Remote Systems*