Application Report
SPRA745A - June 2001

Migrating CCS 1.20/CCS 1.0 Projects to CCS 2.0

Ki-Soo Lee *Code Composer Studio, Applications Engineering*

ABSTRACT

This application report describes the process of migrating an existing Code Composer Studio™ (CCS) version 1.x (v1.x) project to a CCS v2 project.

Contents

1 Introduction

Code Composer Studio version 2 contains new project management features that improve the usability of the CCS Integrated Development Environment (IDE). Such features include the ability to open multiple projects in the workspace and support for multiple build configurations/targets. To support these new features, both the format and syntax of Code Composer Studio project files have been modified. Also, Code Composer Studio v2 project files use a new file extension (*.pjt) to differentiate them from v1.x project files. Because of these differences, projects from v1.x must be migrated to the v2 project format. This application report shows each step of the process.

If the converted CCS project contains DSP/BIOS™ files, the DSP/BIOS configuration (*.cdb) file must be converted separately. Also, the Chip Support Library (CSL) has undergone some major changes since version 1.2. Both topics are discussed in this document.

NOTE: The TMS320C6000™ (C6000™) examples in this document show a migration from CCS v1.2 to CCS v2.0 for building on a C6711 DSP starter kit (DSK). The migration process is similar for the TMS320CC5000™ (C5000™) platform.

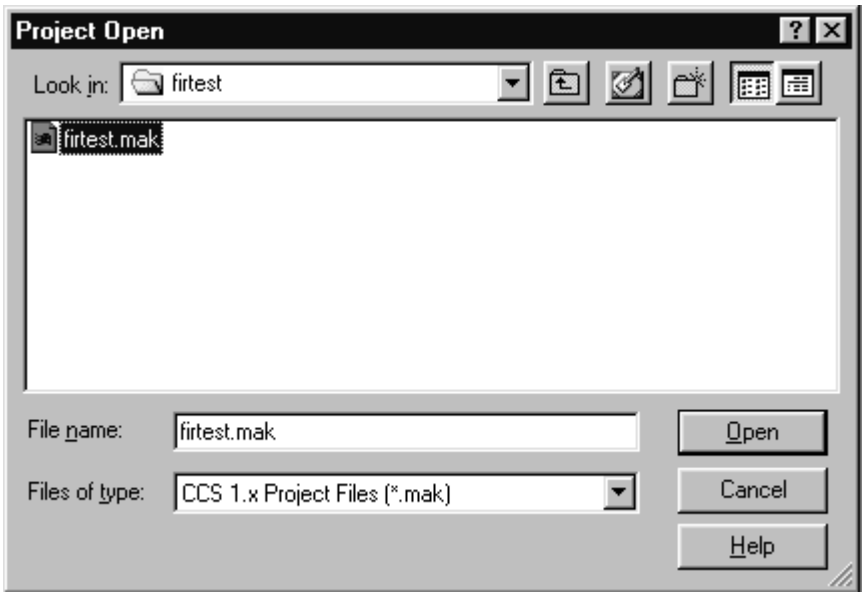
2 Converting a v1.x Project File to a v2 Project File

2.1 Converting a *.mak File to a *.pjt File

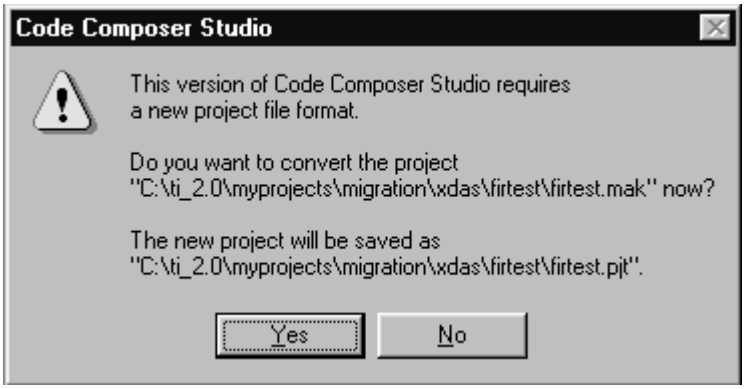
When you open a v1.x project file in Code Composer Studio v2, a new project file is automatically created. The new project file (*.pjt) is created in the same directory. The old project file (*.mak) remains unchanged. In this example, we will open a CCS v1.2 project file called firtest.mak and convert it into a CCS v2.0 project file (firtest.pjt). Firtest.mak is the project file for the FIR example that comes with CCS v1.2. It can initially be found in the following path where CCS v1.x is installed: *<install path>\<target family>\examples\xdas\firtest,* but a copy of the project folder is also provided with this document. For this demonstration, the project folder is in *C:\ti_2.0\myprojects\migration\xdas\firtest\..*

NOTE: When a path name is given, <install path> will be used to refer to the path where CCS v2 is installed. The default install path name is *C:\ti.* In this example, the default install path of *C:\ti* is not used when installing CCS v2, *C:\ti_2.0* is used instead. Please be aware of this when full path names are shown in screen shots. Also, this example uses C6000 CCS v2, so the target platform is c6000. However, this example can also be applied to C5000 CCS v2. Simply replace references to c6000 in the path name with c5000.

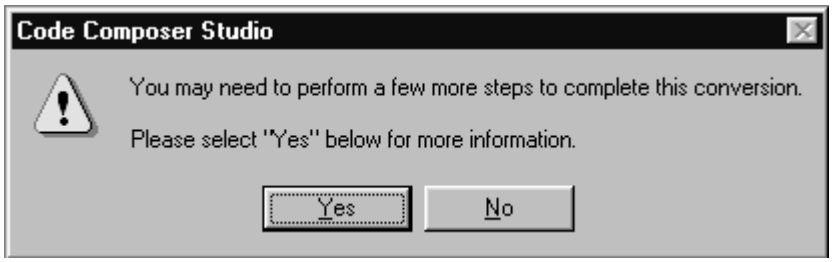
1. Select the command Project–>Open.
2. In the Files of type drop-down list, select CCS 1.x Project Files (*.mak) to search only for v1.x project files. Then browse to the location of the firtest.mak file.



3. Click Open.
4. A prompt informs you that the project is stored in an old format and asks if you want to convert the project to the new format.



5. Click Yes.
6. A prompt informs you that more steps may be necessary for CCS to completely migrate the project successfully. This is usually referring to the DSP/BIOS configuration file, which needs to be converted separately (see section 3 of this document) in addition to other migration issues covered in section 2.2. If the Chip Support Library (CSL) is being used, section 4 describes what needs to be done to convert a v1.x project using CSL to v2.



7. Click Yes. This brings up the on-line help documentation for migrating v1.x projects to v2. Close the help documentation for now.
8. The project file has been successfully migrated to a v2 project file and the new project file is saved and renamed firtest.pjt.
9. The project file has been successfully converted to a v2 project file; however, firtest.pjt also contains a DSP/BIOS configuration file (firtest.cdb) that needs to be migrated to v2 before the project can build properly. Section 3 covers how this is done.

During project conversion, a Code Composer Studio dialog box may inform you that a particular file cannot be found at the location specified in the v1.x project file. If the file that cannot be found is one of the DSP/BIOS generated files (*cfg.h*, *cfg.s*), then click the button labeled Ignore in the Code Composer Studio dialog box. The file will be regenerated when you convert the DSP/BIOS configuration file.

Otherwise, specify the correct location of the file by clicking on the Browse button, specifying the correct location of the file in the Open dialog file, and clicking Open.

Please note that the above error should not occur during the migration of firtest.pjt but was created to show an example of what the error looks like for purposes of demonstration.

NOTE: C6000 Code Composer Studio v1.x contained separate runtime libraries for C (*rts6*.lib*) and C++ (*rtscpp6*.lib*). In C6000 Code Composer Studio v2, functions for both C and C++ are provided in the same runtime libraries (rts6*.lib). If your C6000 v1.x project specifies a C++ library, you must change the specification. For example, change *rtscpp6200.lib* to *rts6200.lib*. This also applies to C5000 Code Composer Studio v2.

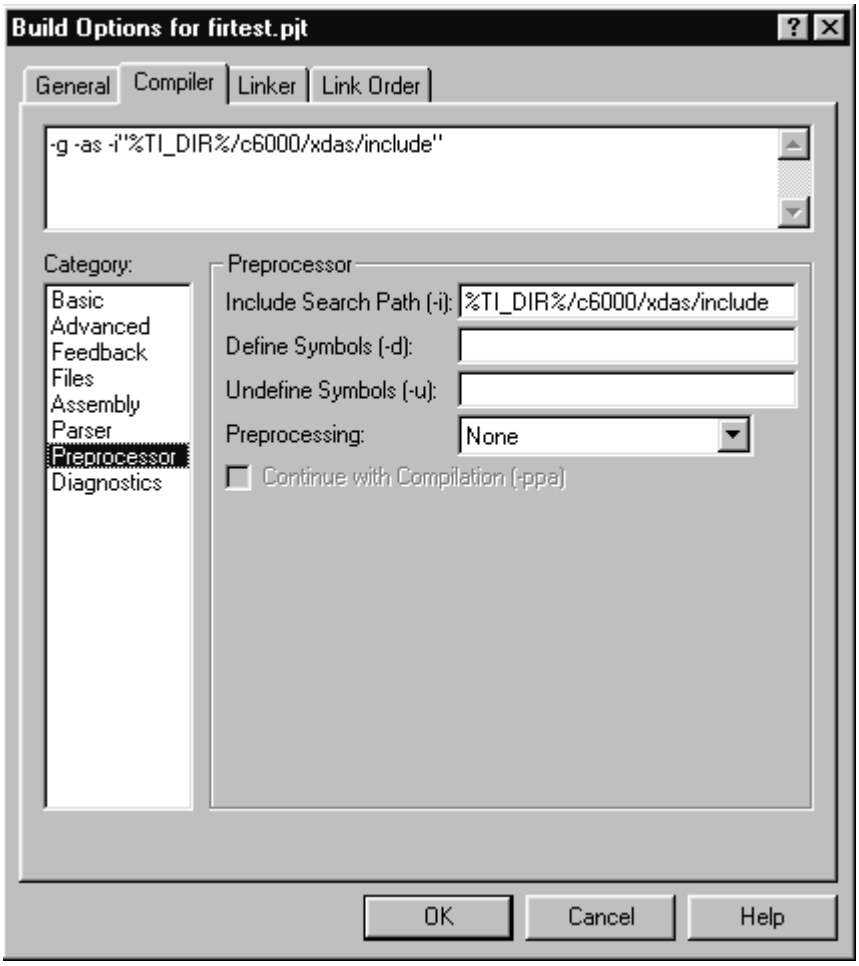
2.2 Migration Issues

Even after a clean v1.x to v2 project conversion, more changes may still need to be made to the project file to ensure a smooth conversion. More complex project files that use additional libraries, have additional include paths, or have other build options enabled may still have some migration issues that need to be resolved. The example project file, firtest.pjt, has some of these problems since it uses additional libraries. The following sections describe what these issues may be, how to resolve them and how to avoid other potential issues.

2.2.1 Project Build Options

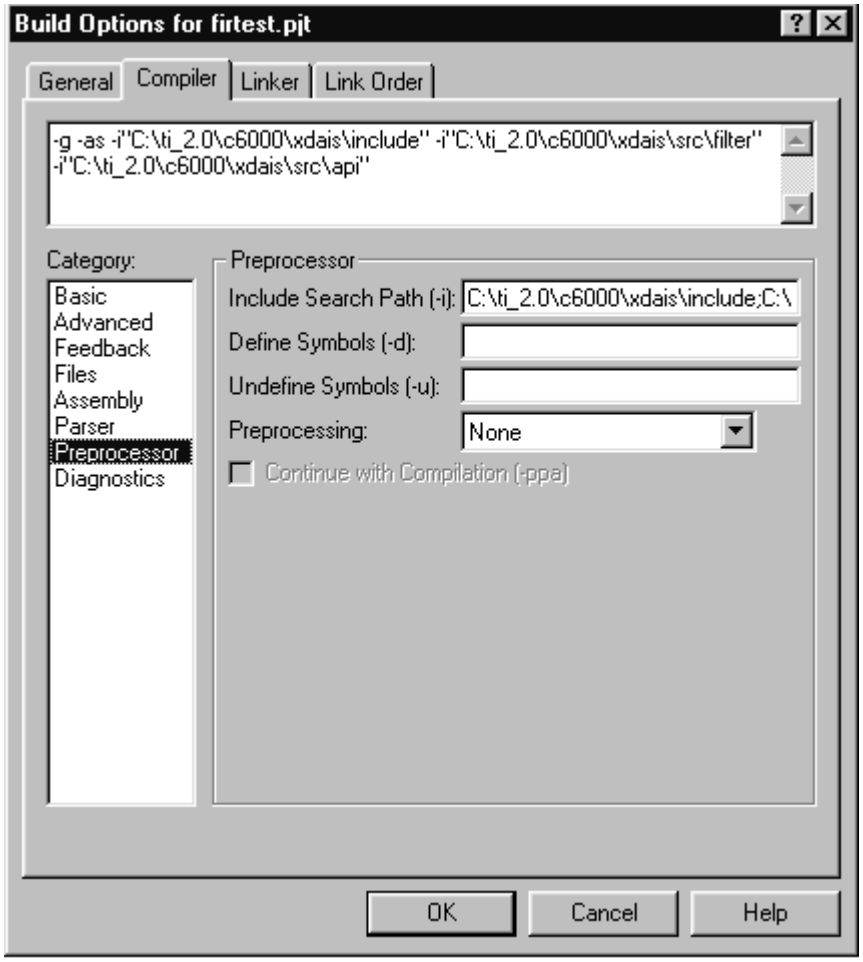
The converted project uses the same build options that were set for your Code Composer Studio v1.x project. The installed directory structure for Code Composer Studio v2 differs from that of v1.x. For example, <*install path*>*c6000**xdas* path in CCS v1.2 has been changed to <*install path*>*c6000**xdais* and all the include header files are no longer in *xdais**include* but grouped together with the associated source file. Also the <*install path*>*ssl**include* and <*install path*>*ssl**lib* paths no longer exist in v2 since the Chip Support Library has been merged with DSP/BIOS (see section 4). If your v1.x project defines search paths in the build options, it may be necessary to update these paths to point to the correct v2 installed directories. For example, if you used the Include Search Path option to specify the location of installed header files, open the Build Options dialog box and specify the path to the desired Include folder in Code Composer Studio v2. For the example, you will first need to edit the include paths to point to the right location.

1. Select Project–>Build Options.
2. In the Compiler dialog, select the Preprocessor category.



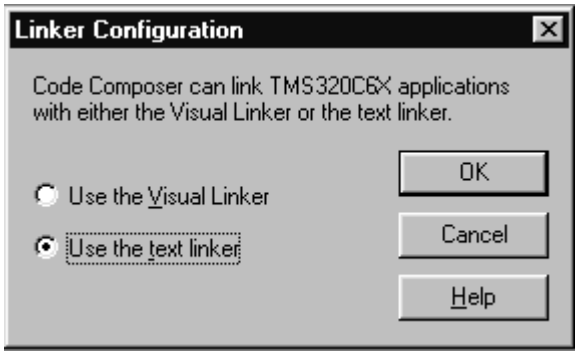
3. Edit the Include Search Path field by replacing the current paths with the following paths:
   <install path>\c6000\xdais\include;<install path>\c6000\xdais\src\api;<install path>\c6000\xdais\src\filter

   NOTE: The *xdas* path in CCS v1.2 has been renamed to *xdais* in CCS v2. In addition, the v1.2 header files that were in *xdas**include* are not all in v2 *xdais**include* directory. Many have been moved to the associated source file or group (for example, fir.h is now in *xdais**src**filter*); therefore, it is necessary to add additional paths to the Include Search Path field in the Build Options.



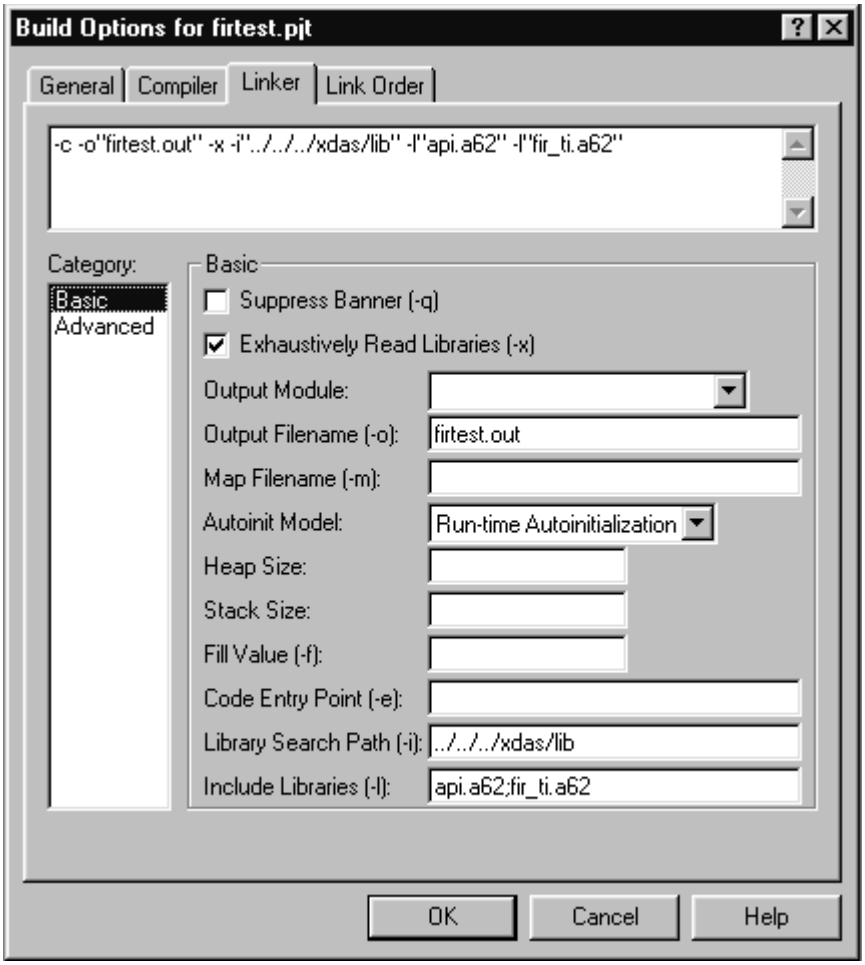
The Compiler options dialog contains additional options that define paths such as Asm Directory and Obj Directory. The Linker options dialog contains those that define Library Search Path and Include Libraries. For the example, firtest.pjt uses xdais libraries where the search paths for these libraries need to be respecified. Also the xdais library names have been slightly changed (ex: *libraryname.a6x* is now *libraryname.l6x*), and must be updated in the build options also. Also note that the assembler tab is no longer a part of the Project Build Options. In CCS v2, the Compiler tab is used to set compiler and assembler options.

NOTE: In this example, the Text Linker (or Legacy Linker) is being used; therefore, we must make sure that we have the Text Linker specified in our environment. You can check this by going to Tools–>Linker Configuration and verifying that the Text Linker option is selected as shown:



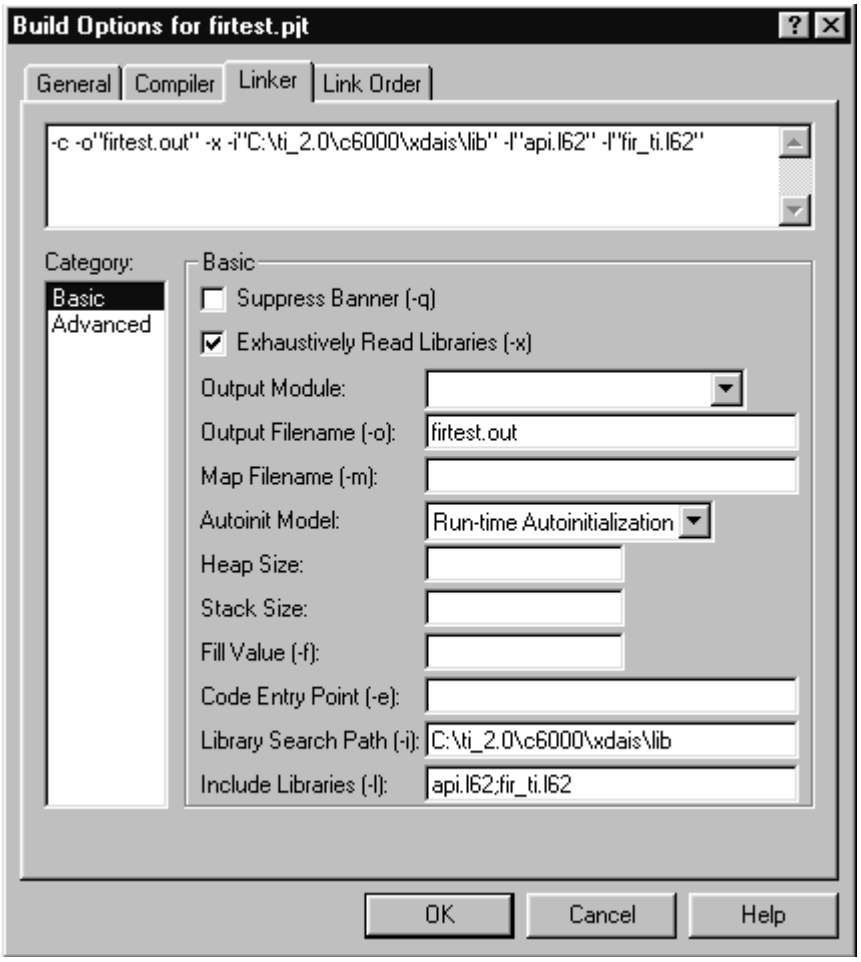
To change the Library Search Path and library names in the Include Libraries field:

1. Select Project–>Build Options. Then select the Linker tab.



2. In the Linker dialog, replace *../../../xdas/lib* with the location of the xdais libraries (found in *<install path>\c6000\xdais\lib)* in the Library Search Path field.
3. Edit the Include Libraries field by replacing *api.a62;fir_ti.a62* with *api.l62;fir_ti.l62.*

   Your Linker dialog should now look like this:



NOTE: Libraries are now also being included by the *cfg.cmd file, which may change the order in which libraries are searched. Because of this, undefined symbol errors may be generated by the linker. If these errors occur, it may be necessary to enable the Exhaustively Read Libraries (–x) linker option.

The Build Options dialogs have been modified in Code Composer Studio v2. For more information on setting build options, see the Code Composer Studio online help: Help>Contents>Using CCS IDE–>Project Environment–>Working with Projects–>Building a Project–>Setting Build Options.

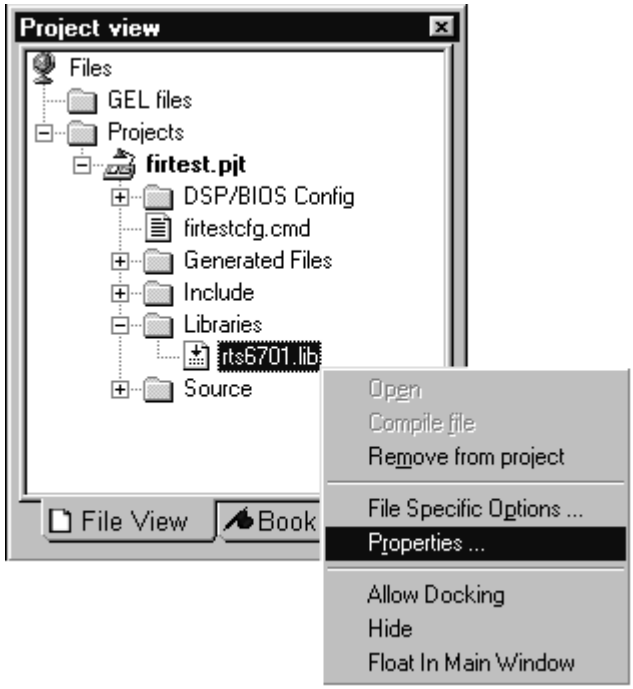
2.2.2 C/C++ Runtime Libraries

One of the key changes with the C/C++ Runtime Libraries is that there is one runtime library (rts*.lib) for both C and C++ functions. The rtscpp.*lib for C++ functions has been eliminated (that is, *rtscpp6200.lib* is now part of *rts6200.lib*).

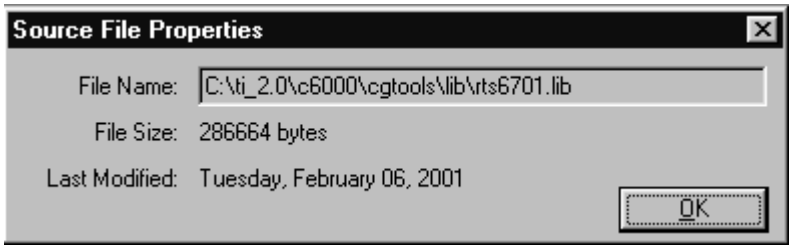
Also, if the converted project contains a C/C++ runtime library and you installed the v1.x and Code Composer Studio v2 products under different paths, it is possible that a v1.x runtime library, is specified for your converted v2 project. Firtest.pjt does not contain a C/C++ runtime library, so this process is not needed for its migration. However, a runtime library was added to the example to demonstrate this migration issue. Please note that the project in this document does **not** have the runtime library shown in the following screen shots (nor is it needed).

The following steps describe how to ensure that the correct version of the runtime library is specified:

1. View the current runtime library specification.
   a. In the Project View, expand the Libraries folder to view the runtime library file (*.lib).
   b. Right-click on the runtime library filename and select Properties.



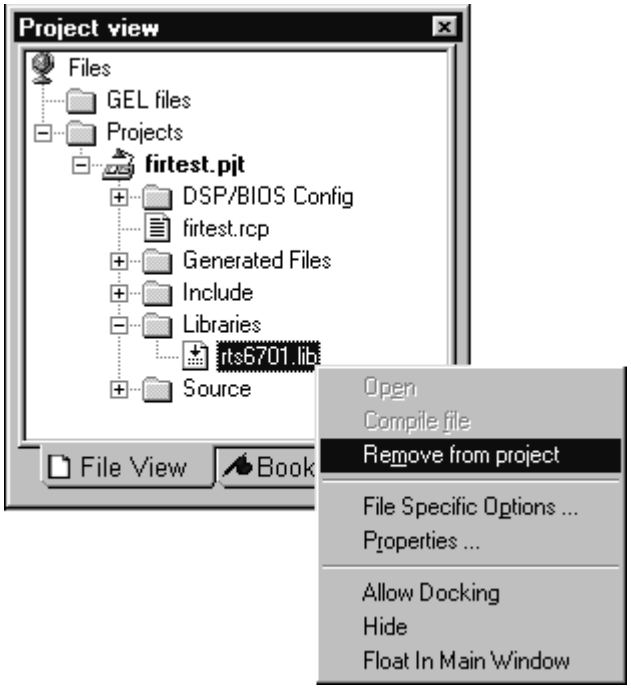
   c. In the Source File Properties dialog box, view the File Name field and verify that the correct path to the Code Composer Studio v2 installation is specified. In nearly all cases, the names of the C/C++ runtime libraries are the same in v1.x and v2; therefore, it is necessary to use the path to verify the correct version.



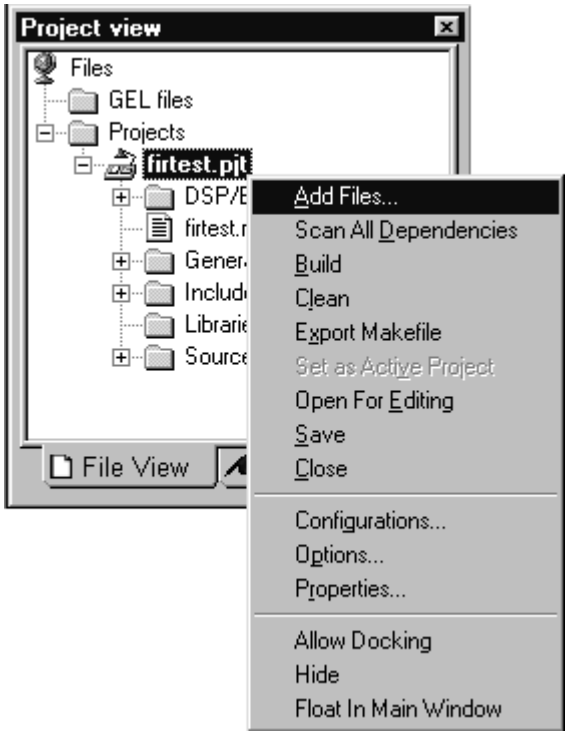
NOTE: If you installed Code Composer Studio v2 in *C:\ti*, the C/C++ runtime libraries are located in *C:\ti\target\cgtools\lib*.lib*, where target represents your target device: c5400, c5500, or c6000.

2. If necessary, change the runtime library specification.

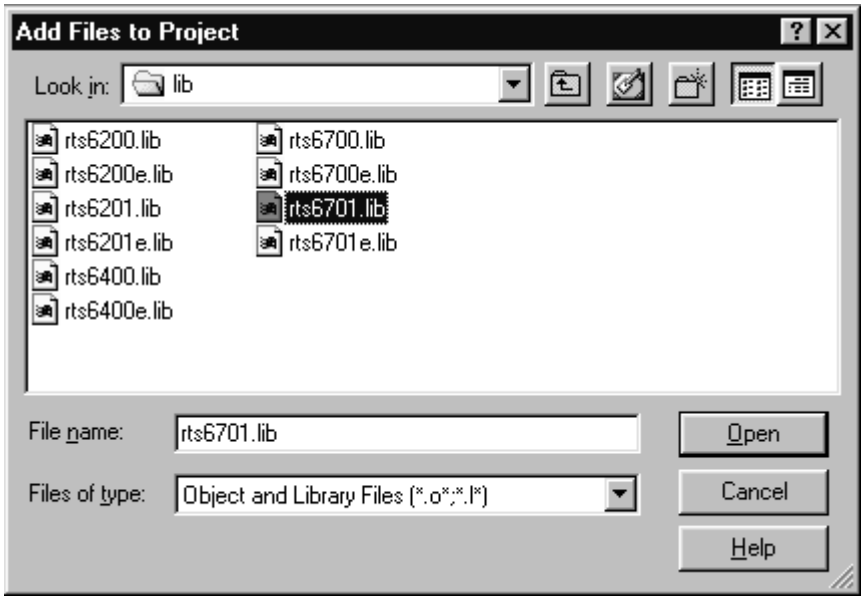
   To specify a new runtime library, you must remove the existing file from the project, and then add the desired file.
   a. In the Project View, expand the Libraries folder to view the runtime library file (*.lib).
   b. To remove the Code Composer Studio v1.x runtime library from the project, right-click on the filename and select Remove from project.



c. To specify the new runtime library, right-click on the project name (*.pjt) and select Add Files.



d. In the Add Files to Project dialog box, browse to the directory that contains the Code Composer Studio v2 runtime libraries. See the above note for additional information on the location of runtime libraries in Code Composer Studio v2.



e. Select the desired runtime library.

f. Click Open.

2.2.3 Include Search Path

The implementation-defined search path that CCS uses to locate header files is changed in CCS v2. If your source code uses the syntax

#include <file.h>

to include a header file that is not located in a CCS installed directory, add the location of the header file to the search path by using the Include Search Path option.

1. Select Project–>Build Options.
2. In the Compiler dialog, select the Preprocessor category.
3. Edit the Include Search Path field.
4. Click OK.

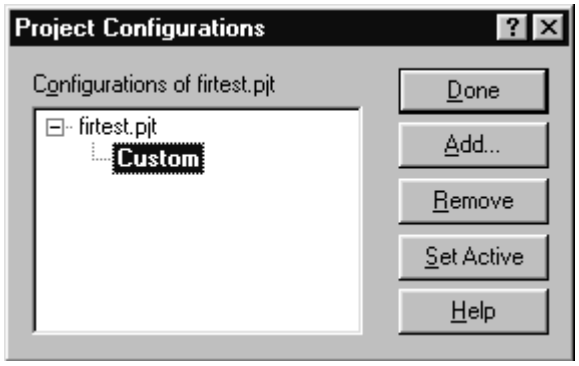
2.2.4 Linker Command File

The linker command file (*.cmd) may contain references to object files and/or object libary files. If those references exist, they must specify the correct paths.

NOTE: The recommended method is to add files to the project (in the Project View, right-click on the project file (*pjt) and choose Add Files), rather than place references in the linker command file.

2.2.5 Project Configuration

Code Composer Studio v2 supports multiple configurations for individual projects. Each project configuration defines a set of build options. This means that within one project file, you can have, for example, a build that includes all debug symbols (*Debug*), a build that has optimization enabled and no debug symbols generated (*Release*), which can also switch between the two configurations. Creating a new project in v2 creates two default configurations: *Debug* and *Release*. Separate Debug and Release directories are created as subdirectories of the directory containing the project file. By default, object files and executables generated during the build process are located in these directories. Converting a v1.x project to a Code Composer Studio v2 project creates a single default project configuration named *Custom*. The location of the *Custom* configuration is defined to be the same directory that contains the converted project file. By default, object files and executables are generated in this directory. A single project configuration helps Code Composer Studio v2 support your existing directory structure and build options. To view all the Project Configurations, select Project–>Configurations.

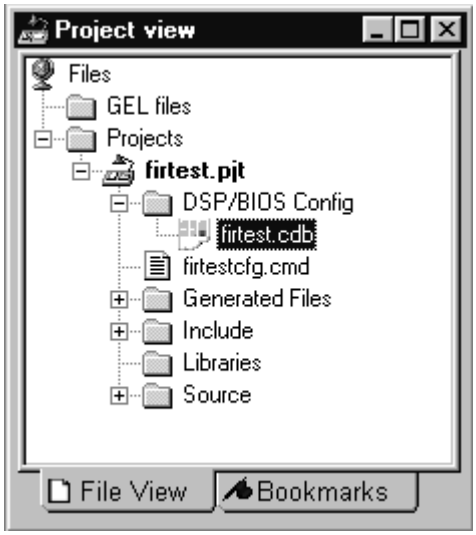


Currently, there is just the one default configuration, *Custom*, created by CCS v2 during the migration. Additional project configurations can be added after conversion. Any added configurations will operate the same as those created with a new project. For more information on project configurations, see Code Composer Studio online help: Help–>Contents–>Using CCS IDE–>Project Environment–>Working with Projects–>Selecting a Project Configuration.

3 Converting a v1.x DSP/BIOS Configuration File to v2 Format

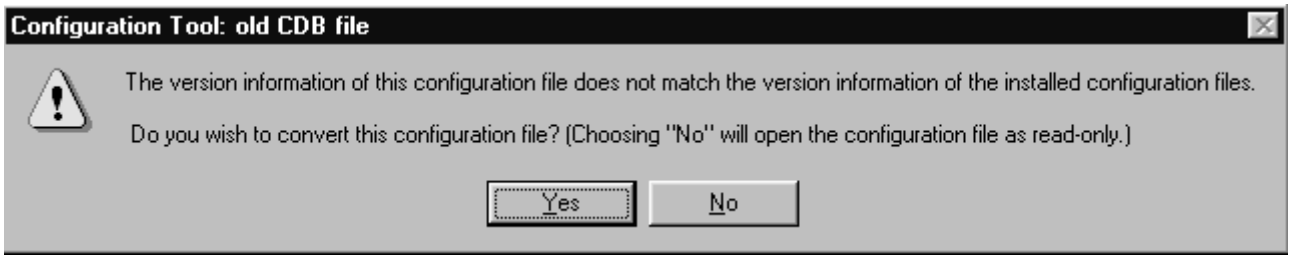
If the converted CCS project contains DSP/BIOS files, the DSP/BIOS configuration file must be converted separately. In firtest.pjt, there is a separate DSP/BIOS configuration file that needs to be converted.

1. In the Project View, expand the DSP/BIOS Config folder to view the DSP/BIOS configuration file (firtest.cdb).



2. To open the DSP/BIOS configuration file, double click on the filename firtest.cdb.

   A prompt informs you that the version information of this configuration file does not match the version information of the installed configuration files and asks if you wish to convert this configuration file.



3. Click Yes. A second prompt informs you that the update succeeded and the original configuration file was renamed.

   Note that a backup copy of the original DSP/BIOS Configuration file is always saved.

4. Click OK. The DSP/BIOS Configuration Tool is opened in the CCS control window.

5. Click the Save button on the Standard toolbar to save your converted DSP/BIOS configuration file.

For each project in CCS v2, the Project View contains a new folder named Generated Files. Source files generated when a DSP/BIOS configuration is saved (*cfg_c.c, ***cfg.s*) are now located in the Generated Files folder instead of the Source folder.

NOTE: When a DSP/BIOS configuration file is saved in CCS v2, five files are generated. For example, with the firtest project, the files generated will be firtestcfg.s62, firtestcfg_c.c, firtestcfg.h, firtestcfg.h62, and firtestcfg.cmd. Generated within *cfg.h are the declarations for the objects created using the DSP/BIOS Configuration Tool. Because of this, you do not need to declare the objects manually in your source code. Also generated in the *_c.c file is the function CSL_cfgInit( ), which is used to initialize CSL components (see section 4).

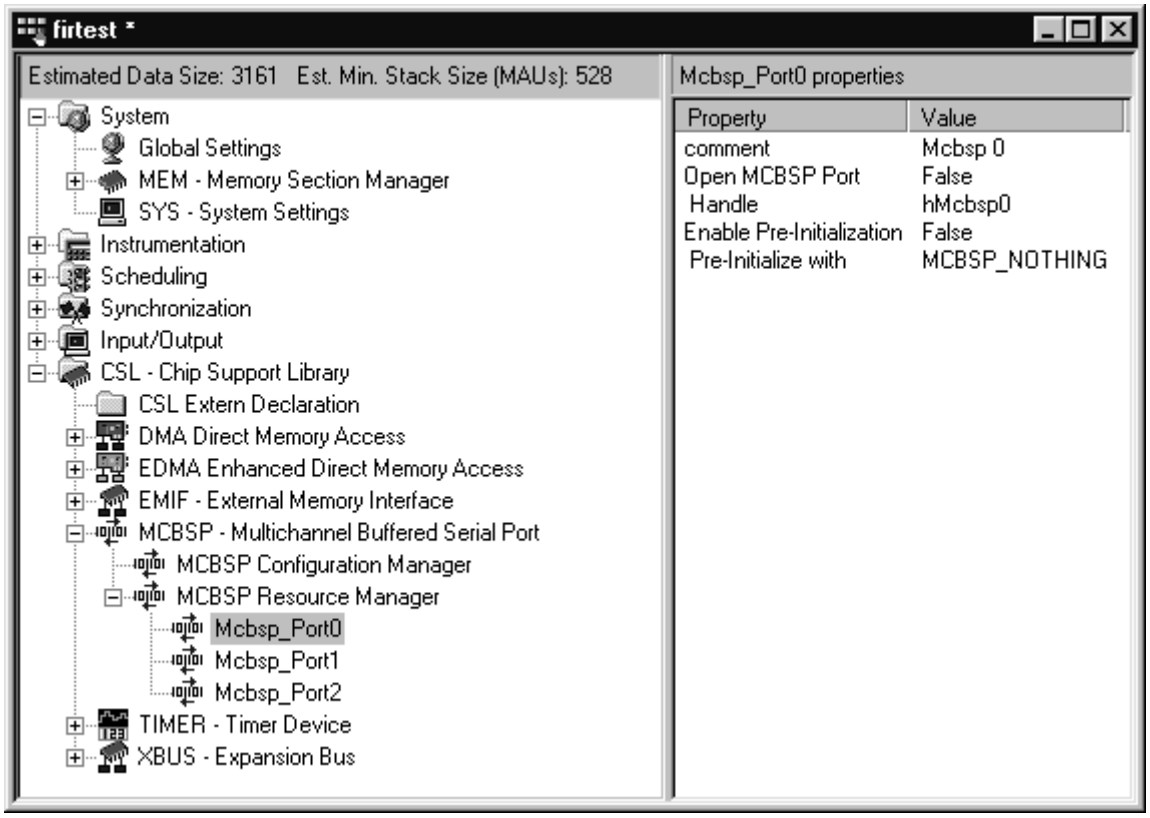
4 Chip Support Library (CSL) Migration Issues

The Chip Support Library (CSL) has undergone changes since version 1.2. These changes make it easier to use and more consistent with other TI software. If the CSL was being used in earlier projects, some modifications must be made to the project/project files for the project to build correctly. Please note that the firtest example does not use any of the CSL components.

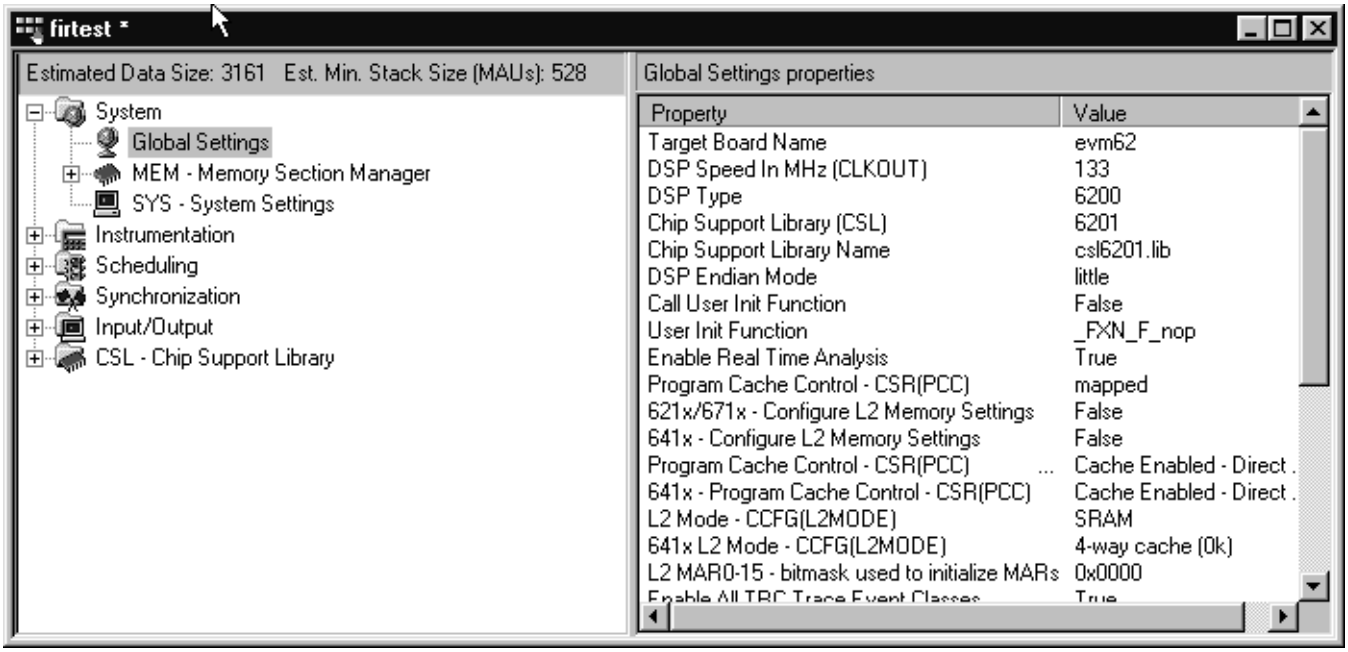
4.1 Migrating From v1.x to v2

The following issues must be addressed before trying to build a project that is ported from CCS v1.x and uses the Chip Support Library.

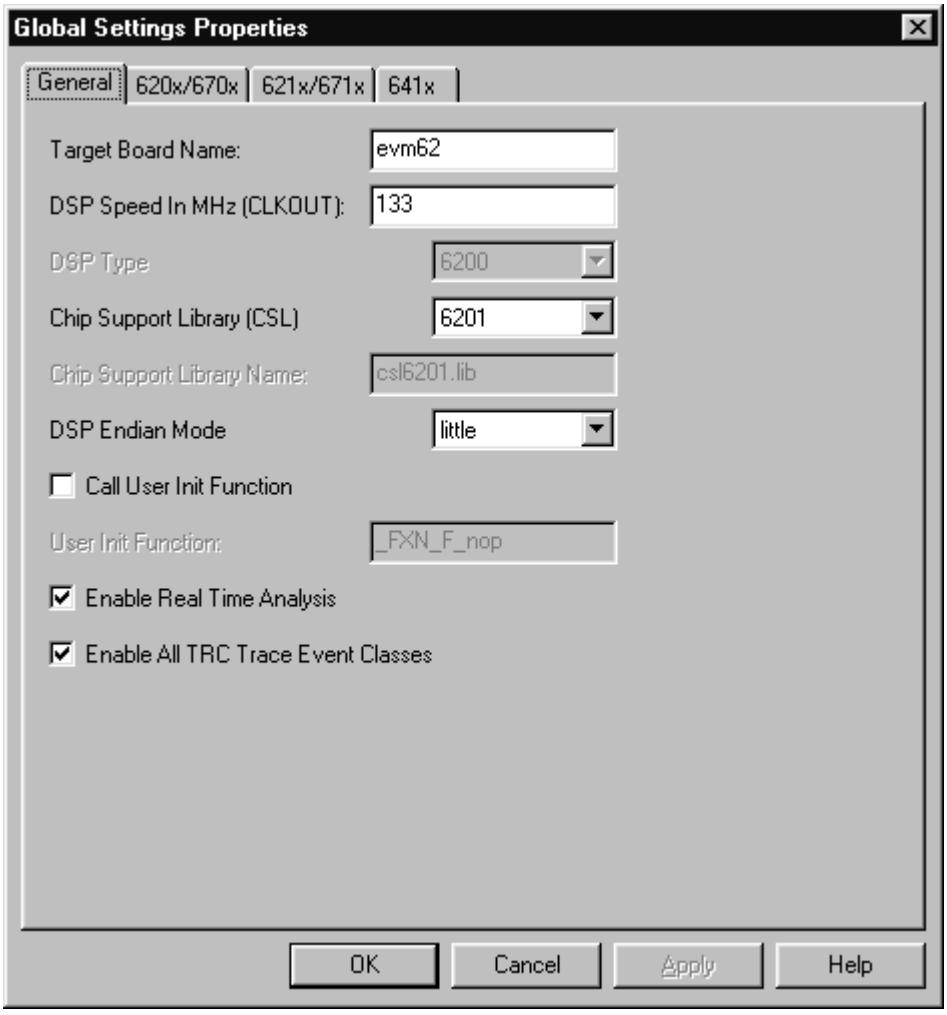
1. The CSL is now tightly integrated with DSP/BIOS. Many of the CSL modules are now configurable using the DSP/BIOS configuration tool.



Also, when a DSP/BIOS configuration is present in the application, DSP/BIOS automatically links the appropriate Chip Support Library. By going to System–>Global Settings in the DSP/BIOS Configuration Tool and looking at the Chip Support Library name field in the right panel, we can see which Chip Support Library will be linked into the application.



The library name can be changed by right-clicking on Global Settings and selecting Properties. This will bring up a dialog box where you can select which Chip Support Library you wish to use.



When a Chip Support Library is selected, DSP/BIOS will automatically link in the selected CSL and also define the CHIP_xxxx global symbol (located in the *.cfg.h header file and generated by the DSP/BIOS Configuration Tool). **If your project contains a DSP/BIOS Configuration file, there should not be any reference to a CSL in the linker build options. Also, if a CHIP_xxxx global symbol definition exists in the compiler build options, it should be removed to avoid redefinition.**

NOTE: If 6211x.lib was part of your v1.x project, then it must be replaced with either csl_6211.lib or csl_6711.lib after a migration to version 2. 6211x.lib does not exist in version 2. This also holds true for the symbol CHIP_6211x, as this symbol does not exist in version 2 and must be replaced with CHIP_6211 or CHIP_6711.

2. The CSL has adopted TI's eXpressDSP™ naming convention for symbols' names.
   PER_funcName()
   PER_varName
   PER_TypeName
   PER_MACRO_NAME

eXpressDSP is a trademark of Texas Instruments.

This means that if you used version 1.20 of the CSL, nearly all symbol names have changed. As an example, what used to be DMA_Start() is now DMA_start(). To lessen the impact of this change, the CSL now has a special header file named *csl_legacy.h* that translates all of the old names into the new names. To make code work that was written using CSL 1.20, the user has to *#include <csl_legacy.h>* in their code before including any of the other CSL header files. This works because in the *csl_legacy.h* header file there are #defines that defines old names to new names, such as:

#define DMA_Start DMA_start

However, even though the legacy header file is included with this release, it is encouraged that users convert their source code to use the new naming convention when referencing CSL identifiers.

3. The 1.2 style HAL (Hardware Abstraction Layer) files are obsolete. Originally, the HAL files were intended only as support files for the API service layer development. However, it was determined that users liked this level of functionality and used it frequently. The 2.0 version of the CSL has a new, revamped set of HAL macros that make them easier to use and are more orthogonal. Again, as with the symbol name changes, the old HAL macros are still accessible by including the *csl_legacy.h* header file. Users are encouraged to modify any use of the old style HAL macros to use the new ones. The new set of HAL macros allows symbolic access to any register, and any bit field, within the registers. A complete reference of these macros is beyond the scope of this document.

4. All CSL header filenames now begin with csl_. This change was made to avoid conflicts with other existing TI and customer header filenames. Some examples of the new filenames are *csl_dma.h, csl_emif.h, csl_mcbsp.h*, etc. Also, all CSL header file references may be removed since these are contained within the header file created by the DSP/BIOS Configuration Tool. Therefore, if code was written using CSL 1.20, all *#include* statements that include CSL header files can simply be removed or must be changed to reflect the new header filenames.

5. The *\ti\c6000\ssl* directory has been eliminated. CSL header files are now located in the DSP/BIOS include directory, and CSL library files are now located in the DSP/BIOS lib directory. This means the compile tools will find these files automatically.

   Users who added the *\ti\ssl\include* and *\ti\ssl\lib* paths to their project settings should remove them.

6. CSL_Init() no longer needs to be called. The function to initialize the CSL is called in the *cfg_c.c source code generated by the DSP/BIOS Configuration Tool. A call to CSL_Init() in your source code should be removed.

7. The dev.6x.lib is no longer being supported in version 2. It has been replaced by the CSL. It is recommended that dev6x.lib users move to using the CSL.

4.2 Enhancements and Changes to CSL

There are other changes and enhancements made to the CSL since v1.2. Although they may not prevent a project from building successfully, the user may find it useful and helpful to be aware of them. Some of these involve stylistic changes that should be followed in later development, and some are enhancements that add to the features of the CSL.

1. Certain cache APIs that take an address as an argument are now of type void*, as opposed to unsigned int. This was done to simplify the use of the APIs because it turned out the majority of the time; the argument was a pointer to a data object. In CSL 1.2, this argument always had to be typecast to an unsigned int.

   CACHE_flush()

   CACHE_clean()

   CACHE_invalidate()

2. For reasons of clarity and future expandability, the configuration functions have been renamed. What was configA is now simply config, and what was configB is now configArgs.
3. The CSL now has big-endian builds as well as little-endian builds. The big-endian library filenames have an ‘e’ at the end, i.e., *csl6201e.lib.*
4. The name of the CSL source archive has been changed from *csl.src* to *csl6000.src*.
5. The CSL may now be used in C++ programs. Although none of the CSL is using C++ extensions directly, the header file declarations now take C++ into consideration.
6. New APIs have been added to the EDMA module making it easier to manage EDMA interrupts and also making it easier to link transfers together. In addition, EDMA reload parameter tables may now be allocated in groups, instead of one-by-one.

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, license, warranty or endorsement thereof.

Reproduction of 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. Representation or reproduction of this information with alteration voids all warranties provided for an associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Resale of TI's 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, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Also see: Standard Terms and Conditions of Sale for Semiconductor Products. www.ti.com/sc/docs/stdterms.htm

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265