Application Report
SPRA913 – June 2003

Creating Portable Projects in Code Composer Studio IDE v2.2

Kaushik Seethapathy
Applications Engineering

ABSTRACT

This application note examines some of the issues with environment variables of Code Composer Studio™ IDE project files in source control and offers techniques on how to resolve them.

Requirements

- Microsoft Windows NT™, Windows™ 98, 2000, or XP Professional.
- Texas Instruments Code Composer Studio for Microsoft Windows (version 2.2 or newer)

Contents

1 Introduction 1
2 Setting Up the Code Composer Studio Project File 2
3 Setting Up the Macro.ini File 2
4 Example Conversion 3

1 Introduction

The Code Composer Studio project manager has many advanced features; however its capability is limited when it comes to porting projects to different users who may have a different root project path. While the majority of source file paths and other information stored in the Code Composer Studio project file is relative to the project file, some information is stored using absolute paths. This makes it very difficult to have a single generic project file that works across many machines and users with different root paths. In Code Composer Studio v2.2, a new feature has been introduced that removes this limitation. Hence, a single generic project file can be used for many different user environments.

Absolute paths in the Code Composer Studio project file can be substituted with a user-defined macro. Users may create as many of these user-defined macros as is required. These macros can point to common files that need to be used when building many versions of the same project. Similarly, Code Composer Studio v2.2 has two built-in macro installation and project paths. The project path macro may be used to remove all dependencies that are project version specific. It can also be used when multiple versions of the same project are located on the user’s machine.

A simple example of the limitation can be illustrated with two users, X and Y, using the same Code Composer Studio project file (Z), that has been placed under version control:

1. X installs Code Composer Studio in a c:\ti1 directory and creates a new project named Z.pjt. X then checks this into source control.
2. Y installs Code Composer Studio in a d:\ti2 directory and checks out Z.
3. Y opens Z.pjt in a text editor and modifies all the path information to d:\ti2 because the project wouldn't run properly.
4. Y checks in all the changes.
5. X checks out the latest version of Z and finds that none of the project settings are the same. Hence, X proceeds to undo all the modifications made since the last check in.
6. X checks in the newly edited Z.

Through this example, it is easy to see that everything will be further complicated with many more users with even more project files. Through a new development to the project settings in Code Composer Studio v2.1, projects can be made portable and the above scenario is no longer a problem with the use of user-defined macros.

Note that this report references from a generic C64xx processor and has Code Composer Studio installed to c:\ti.

2 Setting Up the Code Composer Studio Project File

The only modification that needs to be made to make project files portable is to replace all path information with the `$([keyword])` syntax. Code Composer Studio will recognize this syntax and implement the definition for the keyword in its place as if it were hard-coded. These definitions are to be placed in the macro.ini file, which will be discussed in the following section.

All `[keyword]` variables used must be defined in macro.ini and are case sensitive. Currently the only built-in macro names are `Install_dir` and `Proj_dir`.

3 Setting Up the Macro.ini File

Macros can be used to represent path names in project directories, source files, sub-projects, build settings and commands, and search paths for makefiles. **One important point is that this file must reside in the same directory as projsvr.dll – which is also located in the same folder as cc_app.exe.** Hence, if a user has two versions of Code Composer Studio installed on two different partitions of a hard disk drive (i.e., a non-merge install), multiple macro.ini files must be defined in the appropriate places.

These macros are loaded when cc_app.exe is launched. In turn, any modification to the .ini file would require re-launching Code Composer Studio. This is also the same for timake.exe. Since the macros are global per installation, all projects can use the same set of macros if they are run by cc_app.exe or timake.exe from the same installation directory.

Macro keywords can consist of any alphanumeric character (including underscores but no white-space) and can have only one value. The following is not supported:

```
my_include=c:\common\include;c:\other\include;
my include2=c:\ti\include\;
```

In the case of redefinition, only the first definition of the macro name will be used as all subsequent ones will be ignored:

```
my_include=c:\common\include;
my_include=c:\other\include;
```

One important point to make is that not all built-in macros can be overloaded. For instance, `Install_dir` can be redefined to point to a different location. This would allow advanced users to pick particular system paths from specific installations. `Proj_dir`, however, cannot be overloaded as it points to the .pjt file currently in use. Be sure not to confuse `Proj_dir` (macro) and `ProjectDir` (Code Composer Studio project variable).

Recursive macros can also be defined:

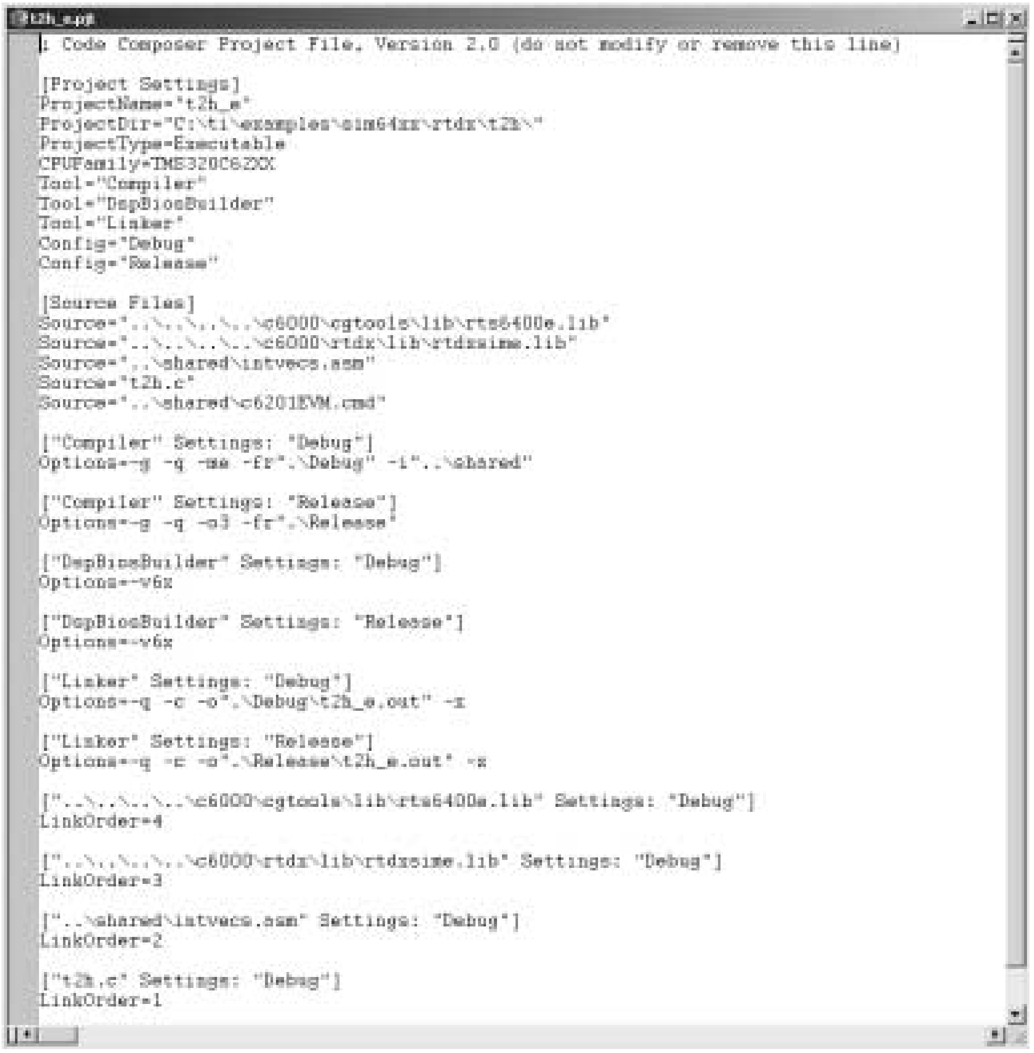
```
cgtools_Dir= c:\ti\c6000\cgtools;
cgtools_LibDir=$(cgtools_Dir)\lib;
```

Paths are the only items that can be made portable and not filenames themselves. Hence, the code, below, is out of scope for Code Composer Studio 2.2:

```
cgtools_Lib=c:\ti\c6000\cgtools\lib \rts6000.lib;
```

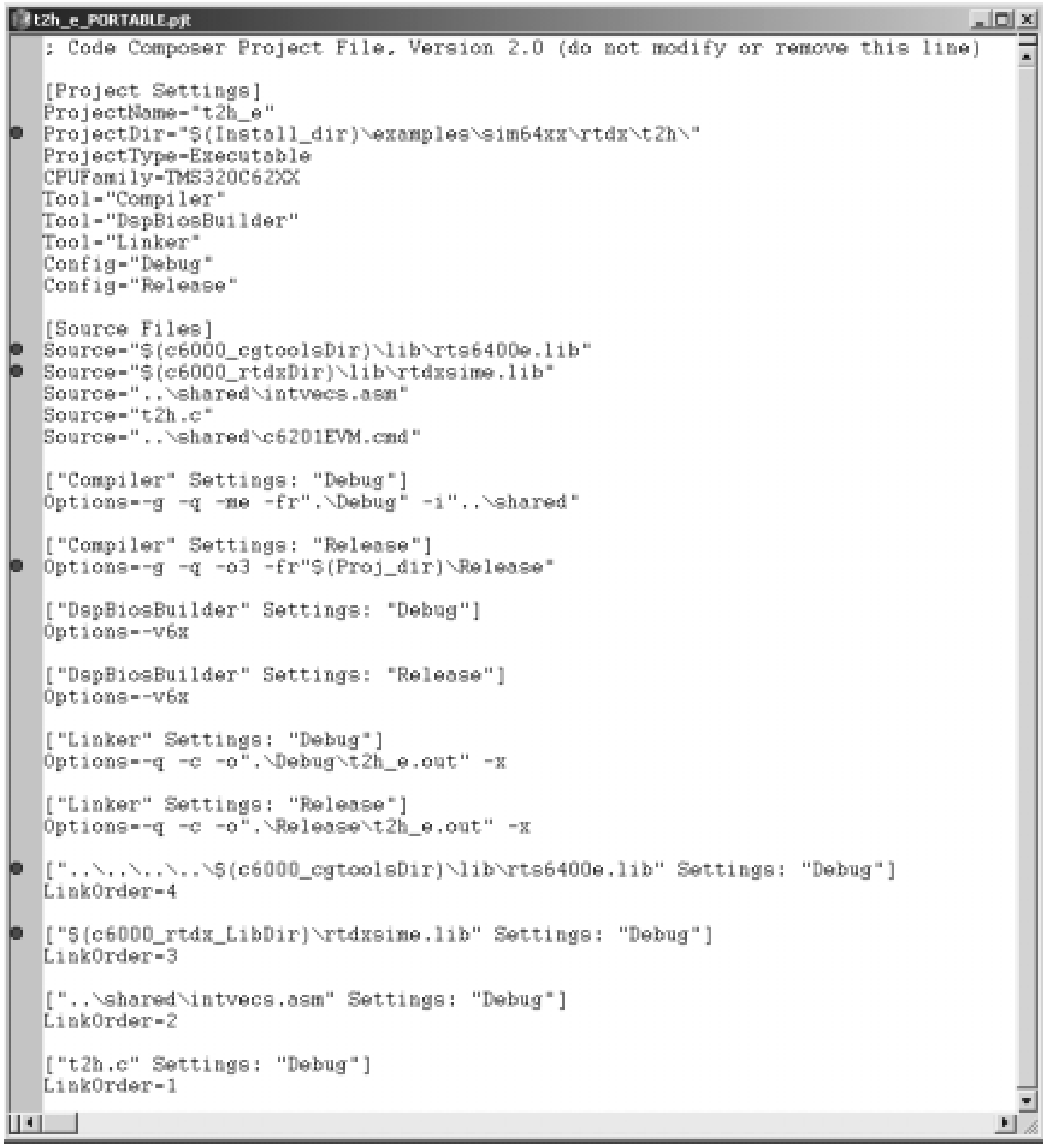
4 Example Conversion

An example of setting up a portable project can be illustrated using t2h.pjt (located in c:\ti\examples\sim64x\rtdx\t2h). The file, as it exists after installation, looks like the following:



```
t2h_e.pjt

; Code Composer Project File, Version 2.0 (do not modify or remove this line)

[Project Settings]
ProjectName="t2h_e"
ProjectDir="C:\ti\examples\sim64xx\rtdx\t2h\"
ProjectType=Executable
CPUFamily=TMS320C62XX
Tool="Compiler"
Tool="DspBiosBuilder"
Tool="Linker"
Config="Debug"
Config="Release"

[Source Files]
Source="..\..\..\..\c6000\cgtools\lib\rts6400e.lib"
Source="..\..\..\..\c6000\rtdx\lib\rtdxsime.lib"
Source="..\shared\intvecs.asm"
Source="t2h.c"
Source="..\shared\c6201EVM.cmd"

["Compiler" Settings: "Debug"]
Options=-g -q -me -fr".\Debug" -i"..\shared"

["Compiler" Settings: "Release"]
Options=-g -q -o3 -fr".\Release"

["DspBiosBuilder" Settings: "Debug"]
Options=-v6x

["DspBiosBuilder" Settings: "Release"]
Options=-v6x

["Linker" Settings: "Debug"]
Options=-q -c -o".\Debug\t2h_e.out" -x

["Linker" Settings: "Release"]
Options=-q -c -o".\Release\t2h_e.out" -x

["..\..\..\..\c6000\cgtools\lib\rts6400e.lib" Settings: "Debug"]
LinkOrder=4

["..\..\..\..\c6000\rtdx\lib\rtdxsime.lib" Settings: "Debug"]
LinkOrder=3

["..\shared\intvecs.asm" Settings: "Debug"]
LinkOrder=2

["t2h.c" Settings: "Debug"]
LinkOrder=1
```

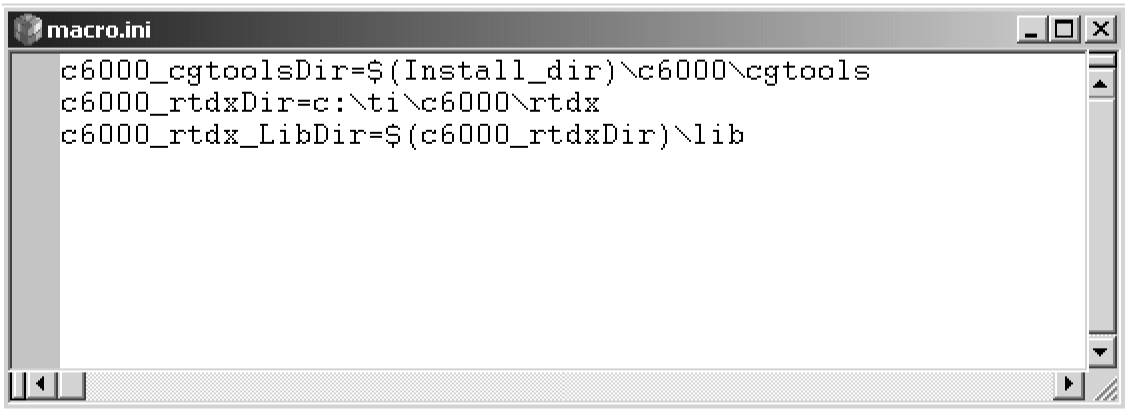
Making this portable involves modifying the some path information that could be hard-coded. This will result in the screenshot below. Note that the breakpoints are used only to indicate the lines that were modified:



t2h_e_PORTABLE.pjt

```
; Code Composer Project File, Version 2.0 (do not modify or remove this line)

[Project Settings]
ProjectName="t2h_e"
ProjectDir="$(Install_dir)\examples\sim64xx\rtdx\t2h\"
ProjectType=Executable
CPUFamily=TMS320C62XX
Tool="Compiler"
Tool="DspBiosBuilder"
Tool="Linker"
Config="Debug"
Config="Release"

[Source Files]
Source="$(c6000_cgtoolsDir)\lib\rts6400e.lib"
Source="$(c6000_rtdxDir)\lib\rtdxsime.lib"
Source="..\shared\intvecs.asm"
Source="t2h.c"
Source="..\shared\c6201EVM.cmd"

["Compiler" Settings: "Debug"]
Options=-g -q -me -fr".\Debug" -i"..\shared"

["Compiler" Settings: "Release"]
Options=-g -q -o3 -fr"$(Proj_dir)\Release"

["DspBiosBuilder" Settings: "Debug"]
Options=-v6x

["DspBiosBuilder" Settings: "Release"]
Options=-v6x

["Linker" Settings: "Debug"]
Options=-q -c -o".\Debug\t2h_e.out" -x

["Linker" Settings: "Release"]
Options=-q -c -o".\Release\t2h_e.out" -x

["..\..\..\..\$(c6000_cgtoolsDir)\lib\rts6400e.lib" Settings: "Debug"]
LinkOrder=4

["$(c6000_rtdx_LibDir)\rtdxsime.lib" Settings: "Debug"]
LinkOrder=3

["..\shared\intvecs.asm" Settings: "Debug"]
LinkOrder=2

["t2h.c" Settings: "Debug"]
LinkOrder=1
```

The corresponding macro.ini file would look something like this:



```
c6000_cgtoolsDir=$(Install_dir)\c6000\cgtools
c6000_rtdxDir=c:\ti\c6000\rtdx
c6000_rtdx_LibDir=$(c6000_rtdxDir)\lib
```