MSP430 GCC

User's Guide

Literature Number: SLAU646A
September 2015–Revised March 2016

Contents

List of Figures

List of Tables

User's Guide
SLAU646A–September 2015–Revised March 2016

MSP430 GCC

Preface: Read This First

How to Use This User's Guide

This manual describes only the setup and basic operation of the MSP430™ GCC compiler and the software development environment but does not fully describe the MSP430 GCC compiler or MSP430 microcontrollers or the complete development software and hardware systems. For details on these items, see the appropriate TI documents listed in Related Documentation From Texas Instruments.

This manual applies to the use of MSP430 GCC as stand-alone package or within the Code Composer Studio™ (CCS) IDE v6.1 and with the TI MSP-FET, MSP-FET430UIF, eZ-FET, and eZ430 development tools series.

These tools contain the most up-to-date materials available at the time of packaging. For the latest materials (including data sheets, user's guides, software, and application information), visit the TI MSP430 website at http://www.ti.com/msp430 or contact your local TI sales office.

Information About Cautions and Warnings

This document may contain cautions and warnings. The information in a caution or a warning is provided for your protection. Read each caution and warning carefully.

CAUTION

This is an example of a caution statement.

A caution statement describes a situation that could potentially damage your software or equipment.

WARNING

This is an example of a warning statement.

A warning statement describes a situation that could potentially cause harm to you.

Related Documentation From Texas Instruments

The primary sources of MSP430 information are the device-specific data sheets and user's guides. The MSP430 website (http://www.ti.com/msp430) contains the most recent version of these documents.

The GCC documentation can be found at http://www.gnu.org. All related information for the MSP430 GCC compiler is available at http://www.ti.com/tool/msp430-gcc-opensource.

MSP430, Code Composer Studio, E2E, eZ430-Chronos, LaunchPad are trademarks of Texas Instruments.
OS X is a registered trademark of Apple Inc.
Linux is a registered trademark of Linus Torvalds.
Windows is a registered trademark of Microsoft Corp.
All other trademarks are the property of their respective owners.

Documents that describe the Code Composer Studio tools (CCS IDE, assembler, C compiler, linker, and librarian) can be found at http://www.ti.com/tool/ccstudio. A CCS-specific Wiki page (FAQ) is available at http://processors.wiki.ti.com/index.php/Category:CCS, and the TI E2E™ Community support forums at http://e2e.ti.com provides additional help.

MSP430 GCC documentation

Using the GNU Compiler Collection, Richard M. Stallman (http://gcc.gnu.org/onlinedocs/gcc.pdf). Refer to the *MSP430 Options* section.

GDB: The GNU Project Debugger, Free Software Foundation, Inc. (https://sourceware.org/gdb/current/onlinedocs/)

Red Hat GCC for MSP430™ Microcontrollers Quick Start Guide (SLAU591)

Calling Convention and ABI Changes in MSP GCC (SLAA664)

CCS v6.1 documentation

MSP430™ Assembly Language Tools User's Guide (SLAU131)

MSP430™ Optimizing C/C++ Compiler User's Guide (SLAU132)

Code Composer Studio™ v6.1 for MSP430™ User's Guide (SLAU157)

MSP430 development tools documentation

MSP430™ Hardware Tools User's Guide (SLAU278)

eZ430-F2013 Development Tool User's Guide (SLAU176)

eZ430-RF2480 User's Guide (SWRA176)

eZ430-RF2500 Development Tool User's Guide (SLAU227)

eZ430-RF2500-SEH Development Tool User's Guide (SLAU273)

eZ430-Chronos™ Development Tool User's Guide (SLAU292)

MSP-EXP430G2 LaunchPad™ Experimenter Board User's Guide (SLAU318)

Advanced Debugging Using the Enhanced Emulation Module (EEM) With Code Composer Studio Version 6 (SLAA393)

MSP430 device data sheets

MSP430 device family user's guides

MSP430x1xx Family User's Guide (SLAU049)

MSP430x2xx Family User's Guide (SLAU144)

MSP430x3xx Family User's Guide (SLAU012)

MSP430x4xx Family User's Guide (SLAU056)

MSP430x5xx and MSP430x6xx Family User's Guide (SLAU208)

MSP430FR4xx and MSP430FR2xx Family User's Guide (SLAU445)

MSP430FR57xx Family User's Guide (SLAU272)

MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Family User's Guide (SLAU367)

If You Need Assistance

Support for the MSP430 devices and the hardware development tools is provided by the TI Product Information Center (PIC). Contact information for the PIC can be found on the TI website at http://www.ti.com/support. The TI E2E Community support forums for the MSP430 at http://e2e.ti.com provide open interaction with peer engineers, TI engineers, and other experts. Additional device-specific information can be found on the MSP430 website at http://www.ti.com/msp430.

1 Introduction

TI has partnered with Red Hat to bring you a new and fully supported open source compiler as the successor to the community driven MSPGCC. The MSP430 GCC uses the MSP430 ABI and is compatible with the TI compiler. This free GCC 4.9 compiler supports all MSP430 devices and has no code size limit. In addition, this compiler can be used as a stand-alone package or used within CCS v6.0 or later. Get started today in Windows, Linux, or Mac environments.

Table 1 gives a quick comparison of the MSP430 TI and GCC compilers.

Table 1. MSP430 TI and GCC Compilers Comparison

Compiler	Proprietary TI Compiler	MSP430 GCC	MSPGCC
Code Size and Performance	✓✓✓	✓	✓
ABI	TI	TI	Community
Integrated in CCS	✓	✓[1]	✗
Stand-alone	✗	✓	✓
Support	TI	TI	Community
Cost Free	✗	✓	✓

[1] The combination of CCS+GCC is completely free of charge with no code size limit.

The MSP430 GCC supports the following:

- MSP430 CPU 16-bit architecture
- MSP430 CPUX 20-bit architecture
- MSP430 CPUXv2 20-bit architecture
- Code and data placement in the lower (<64K) and upper (>64K) memory areas and across the memory boundary
- The hardware multiplier of the MSP430 microcontrollers

This manual describes the use of the MSP430 GCC compiler with the MSP430 ultra-low-power microcontrollers. The MSP430 GCC compiler can be used within CCS version 6.0 or later, or it can be used as a stand-alone package. The compiler supports Windows®, Linux®, and Mac OS X® operating systems. This manual describes only CCS for Windows operating systems. The versions of CCS for Linux and OS X operating systems are similar and, therefore, are not described separately.

2 Installing MSP430 GCC Compiler

MSP430 GCC supports Windows, Linux, and Mac:

- Windows XP 32 bit or 64 bit
- Windows 7 32 bit or 64 bit
- Windows 8 32 bit or 64 bit
- Linux 32 bit or 64 bit
- OS X 64 bit

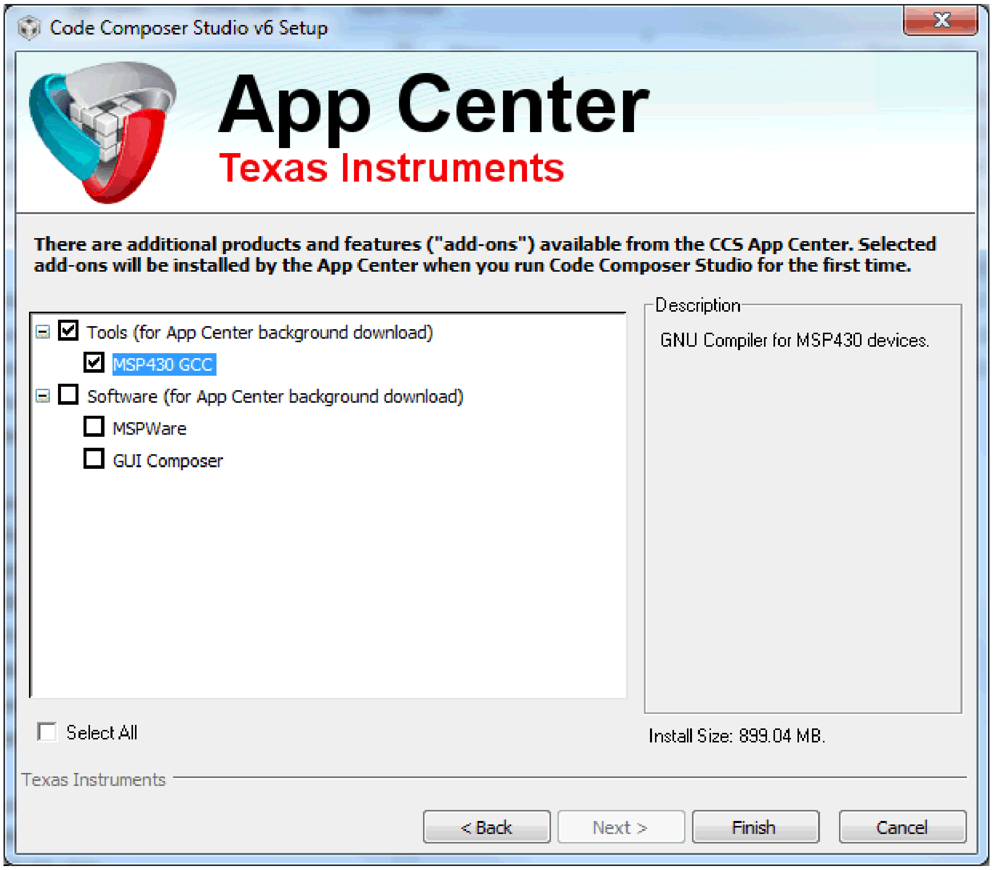
The MSP430 GCC can be installed using two methods.

1. MSP430 GCC can be installed within Code Composer Studio version 6.0 and higher. The MSP430 GCC (compiler only) is available in CCS Apps Center. The corresponding MSP430 GCC support files (header and linkers) are downloaded with a standard emulation package. For details, see Section 2.1.
2. MSP430 GCC can be also downloaded as stand-alone package from the TI website at http://www.ti.com/tool/msp430-gcc-opensource. For details, see Section 2.2.

2.1 Installing MSP430 GCC in CCS

MSP430 GCC compiler can be installed within CCS v6.0 or higher in two ways:

1. During the install process of CCSv6.0, select the MSP430 GCC compiler to be installed as an "add-on" (see Figure 1). MSP430 GCC is installed when CCS runs the first time (see Figure 2).



Code Composer Studio v6 Setup

App Center
Texas Instruments

There are additional products and features ("add-ons") available from the CCS App Center. Selected add-ons will be installed by the App Center when you run Code Composer Studio for the first time.

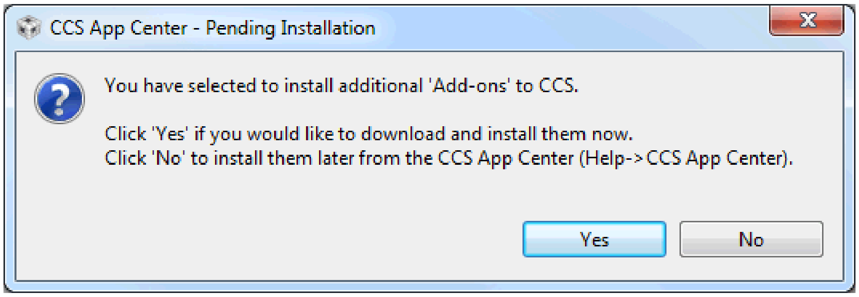
- ☑ Tools (for App Center background download)
  - ☑ MSP430 GCC
- □ Software (for App Center background download)
  - □ MSPWare
  - □ GUI Composer

Description
GNU Compiler for MSP430 devices.

□ Select All

Install Size: 899.04 MB.

Figure 1. MSP430 GCC With CCS Installer



CCS App Center - Pending Installation

You have selected to install additional 'Add-ons' to CCS.

Click 'Yes' if you would like to download and install them now.
Click 'No' to install them later from the CCS App Center (Help->CCS App Center).

Figure 2. MSP430 GCC With CCS Installer

2. If CCS is already installed without MSP430 GCC, MSP430 GCC can be added at a later time through the CCS Apps Center (see Figure 3).
   (a) Go to the menu **View** → **CCS App Center**.
   (b) Select MSP430 GCC
   (c) Click the **Install Software** button to start the installation.

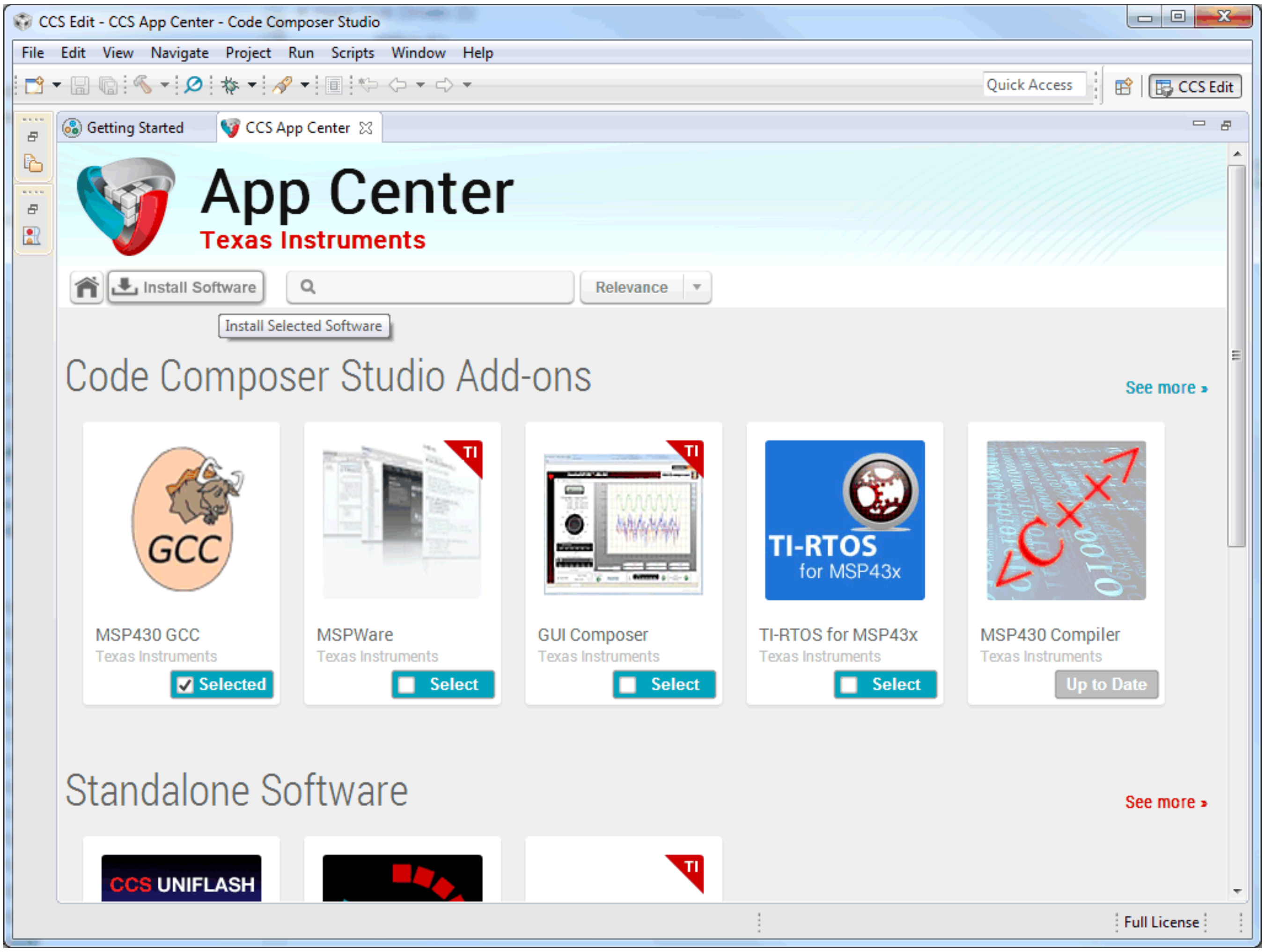


Figure 3. Installing MSP430 GCC Through CCS Apps Center

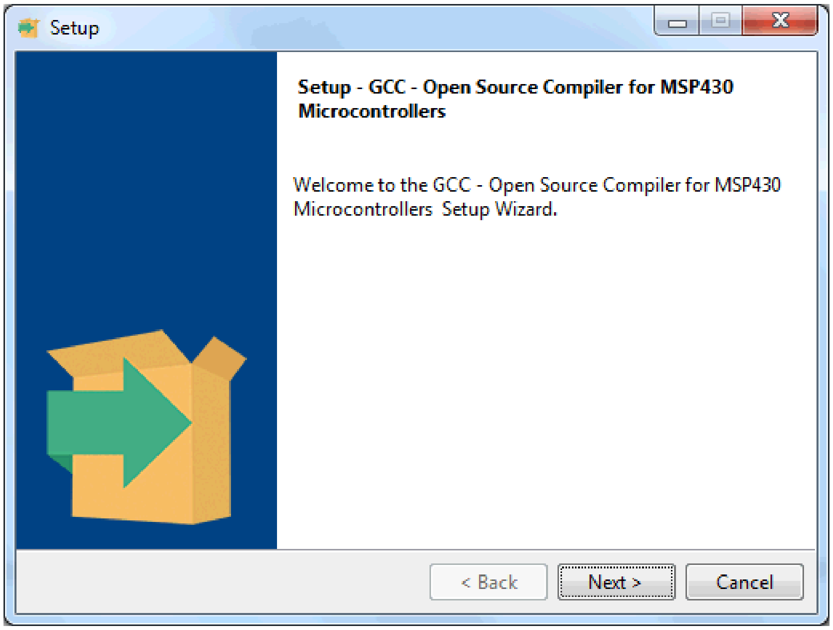
3. After installation is complete, the GCC compiler tools are in this directory within the CCS installation: ccsv6\tools\compiler\gcc_msp430_x.x.x (where xxx denotes the version number).

2.2 Installing MSP430 GCC as Stand-Alone Package

The MGC430 GCC full stand-alone package can be downloaded from the TI website (http://www.ti.com/tool/msp430-gcc-opensource) for all supported operating systems. The MSP430 GCC stand-alone package contains the compiler, device support files, debug stack, and USB drivers.

To install the package:

1. Download the corresponding package installer and run it (see Figure 4).

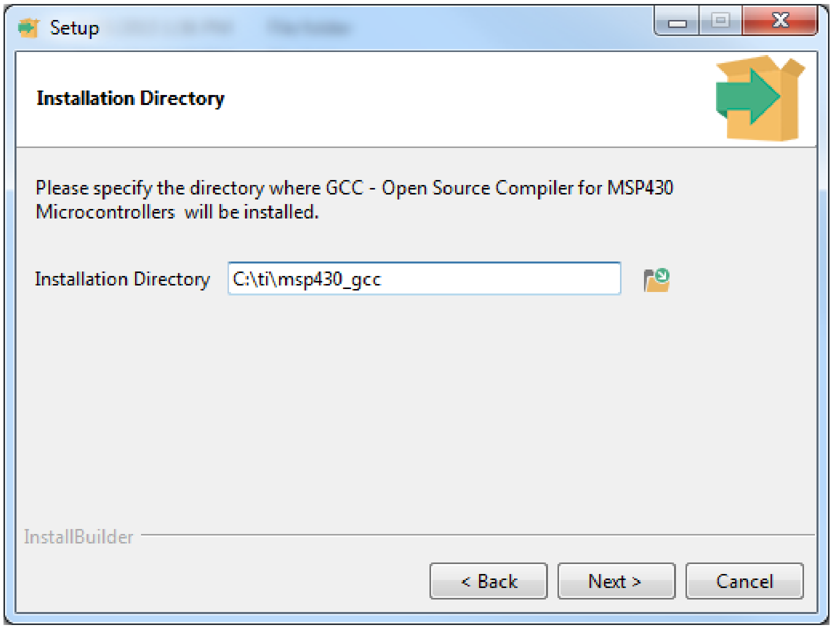


Setup - GCC - Open Source Compiler for MSP430 Microcontrollers

Welcome to the GCC - Open Source Compiler for MSP430 Microcontrollers Setup Wizard.

Figure 4. MSP430 GCC Stand-Alone Package Installer

2. Select the install directory and click **Next** (see Figure 5).



Installation Directory

Please specify the directory where GCC - Open Source Compiler for MSP430 Microcontrollers will be installed.

Installation Directory C:\ti\msp430_gcc

Figure 5. MSP430 GCC Stand-Alone Package Installation Directory

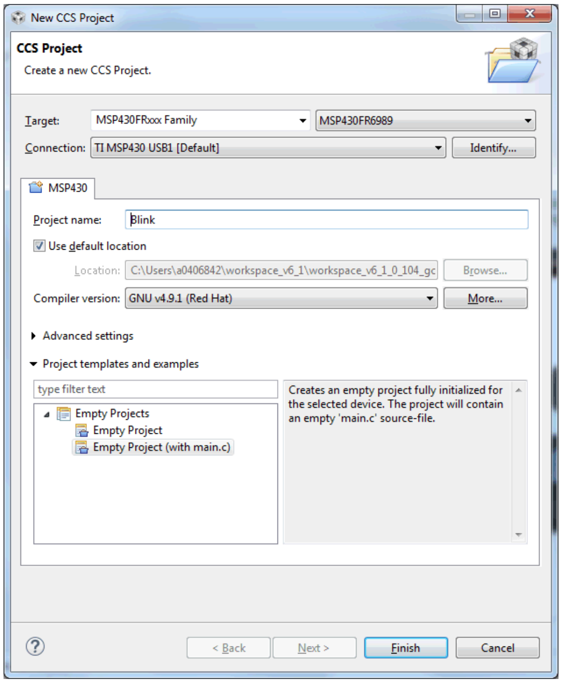
NOTE: For the Linux installer, apply **chmod x** before executing the package.

3 Using MSP430 GCC Within CCS

3.1 Create New Project

This section describes a step-by-step instructions to create an assembly or C project from scratch and to download and run an application on the MSP430 MCU using the MSP430 GCC compiler. Also, the CCS Help presents a more comprehensive overview of the process.

1. Start CCS (**Start** → **All Programs** → **Texas Instruments** → **Code Composer Studio** → **Code Composer Studio**).
2. Create new project (**File** → **New** → **CCS Project**). Select the appropriate MSP430 device variant in the Target field and enter the name for the project.
3. Select **GNU v4.9.1 (Red Hat)** for Compiler version (or any newer version).
4. In the Project template and examples section, select **Empty Project (with main.c)**. For assembly-only projects, select **Empty Project**.



New CCS Project

CCS Project

Create a new CCS Project.

Target: MSP430FRxxx Family | MSP430FR6989

Connection: TI MSP430 USB1 [Default]

MSP430

Project name: Blink

☑ Use default location

Location: C:\Users\a0406842\workspace_v6_1\workspace_v6_1_0_104_gc

Compiler version: GNU v4.9.1 (Red Hat)

Project templates and examples

- Empty Projects
  - Empty Project
  - Empty Project (with main.c)

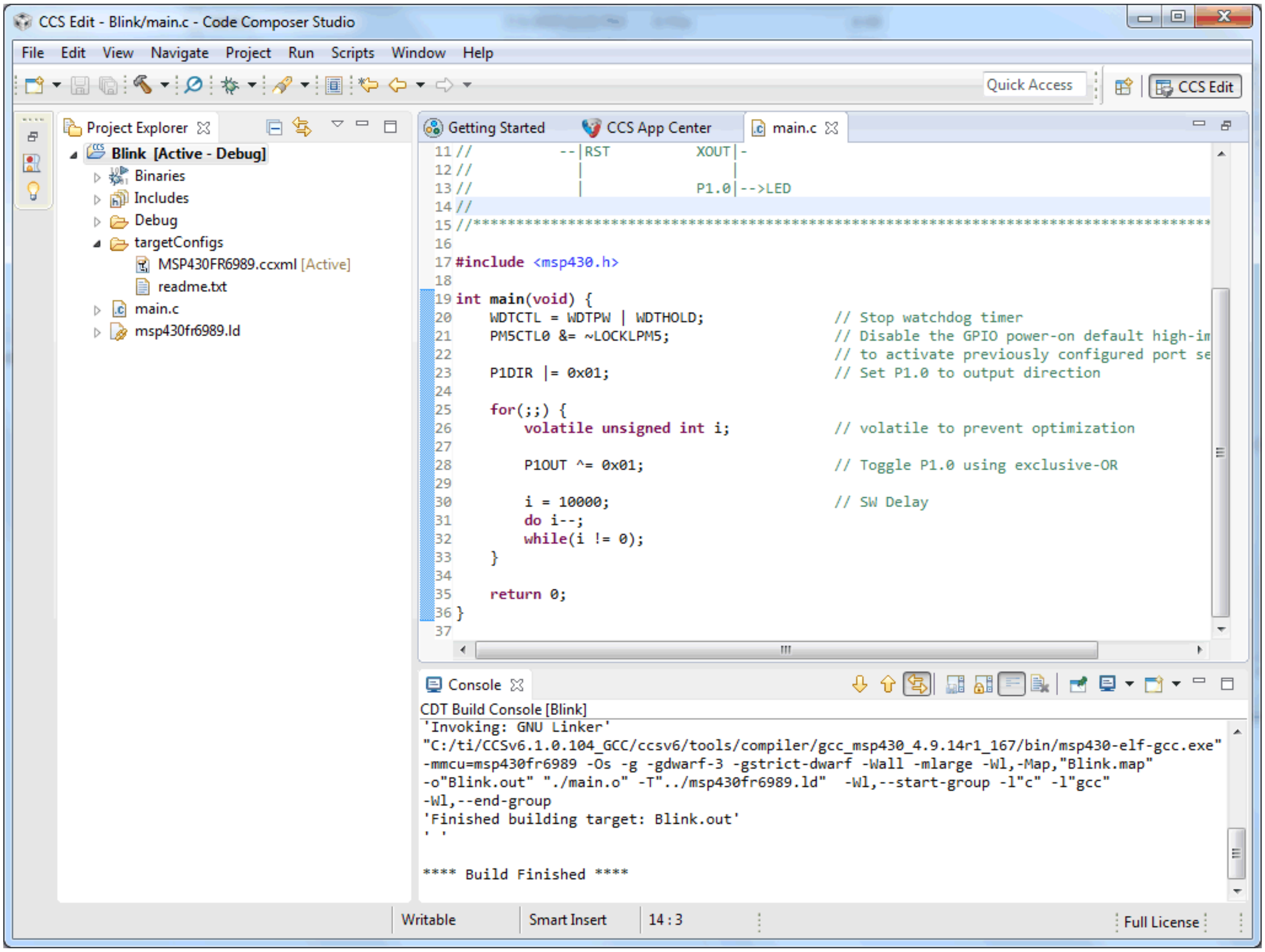
Creates an empty project fully initialized for the selected device. The project will contain an empty 'main.c' source-file.

Figure 6. Creating New CCS Project Using MSP430 GCC

5. If using a USB Flash Emulation Tool such as the MSP-FET, MSP-FET430UIF, eZ-FET, or the eZ430 Development Tool, they should be already configured by default.
6. For C projects, the setup is complete now, main.c is shown, and code can be entered.

   To use an existing source file for the project, click **Project** → **Add Files...** and browse to the file of interest. Single click on the file and click **Open** or double-click on the file name to complete the addition of it into the project folder.
7. Click **Finish** to create a new project that is then visible in the Project Explorer view.

   Notice that the project contains a .ld file (appropriate for the target selected). This is the linker script that contains the memory layout and section allocation. This file is the equivalent of the TI linker command file (.cmd). Now add the necessary source files to the project and build. Similar to TI tools, additional compiler and linker options can be set from **Project Properties**.
8. Enter the program code into the main.c file.
9. Build the project (**Project** → **Build Project**).



```
//            --|RST          XOUT|-
//              |                 |
//              |             P1.0|-->LED
//
//******************************************************************************

#include <msp430.h>

int main(void) {
    WDTCTL = WDTPW | WDTHOLD;                   // Stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5;                       // Disable the GPIO power-on default high-im
                                                // to activate previously configured port se
    P1DIR |= 0x01;                              // Set P1.0 to output direction

    for(;;) {
        volatile unsigned int i;                // volatile to prevent optimization

        P1OUT ^= 0x01;                          // Toggle P1.0 using exclusive-OR

        i = 10000;                              // SW Delay
        do i--;
        while(i != 0);
    }

    return 0;
}
```

CDT Build Console [Blink]

```
'Invoking: GNU Linker'
"C:/ti/CCSv6.1.0.104_GCC/ccsv6/tools/compiler/gcc_msp430_4.9.14r1_167/bin/msp430-elf-gcc.exe"
-mmcu=msp430fr6989 -Os -g -gdwarf-3 -gstrict-dwarf -Wall -mlarge -Wl,-Map,"Blink.map"
-o"Blink.out" "./main.o" -T"../msp430fr6989.ld"  -Wl,--start-group -l"c" -l"gcc"
-Wl,--end-group
'Finished building target: Blink.out'
' '

**** Build Finished ****
```

Figure 7. CCS Project Using MSP430 GCC

10. Debug the application (**Run** → **Debug (F11)**). This starts the debugger, which gains control of the target, erases the target memory, programs the target memory with the application, and resets the target.
11. Click **Run** → **Resume (F8)** to start the application.
12. Click **Run** → **Terminate** to stop the application and to exit the debugger. CCS automatically returns to the C/C++ view (code editor).

3.2 Debug Using MSP-FET, MSPFET430UIF, eZ-FET, eZ430

MSP430 devices can be debugged in CCS using MSP-FET, MSPFET430UIF, eZ-FET, and eZ430 debuggers. For more details, refer to the *Code Composer Studio™ v6.1 for MSP430™ User's Guide* (SLAU157).

3.3 Build Options for MSP430 GCC

The settings required to configure the GCC are numerous and detailed. Most projects can be compiled and debugged with default factory settings.

To access the project settings for the active project, click **Project** → **Properties** for the active project.

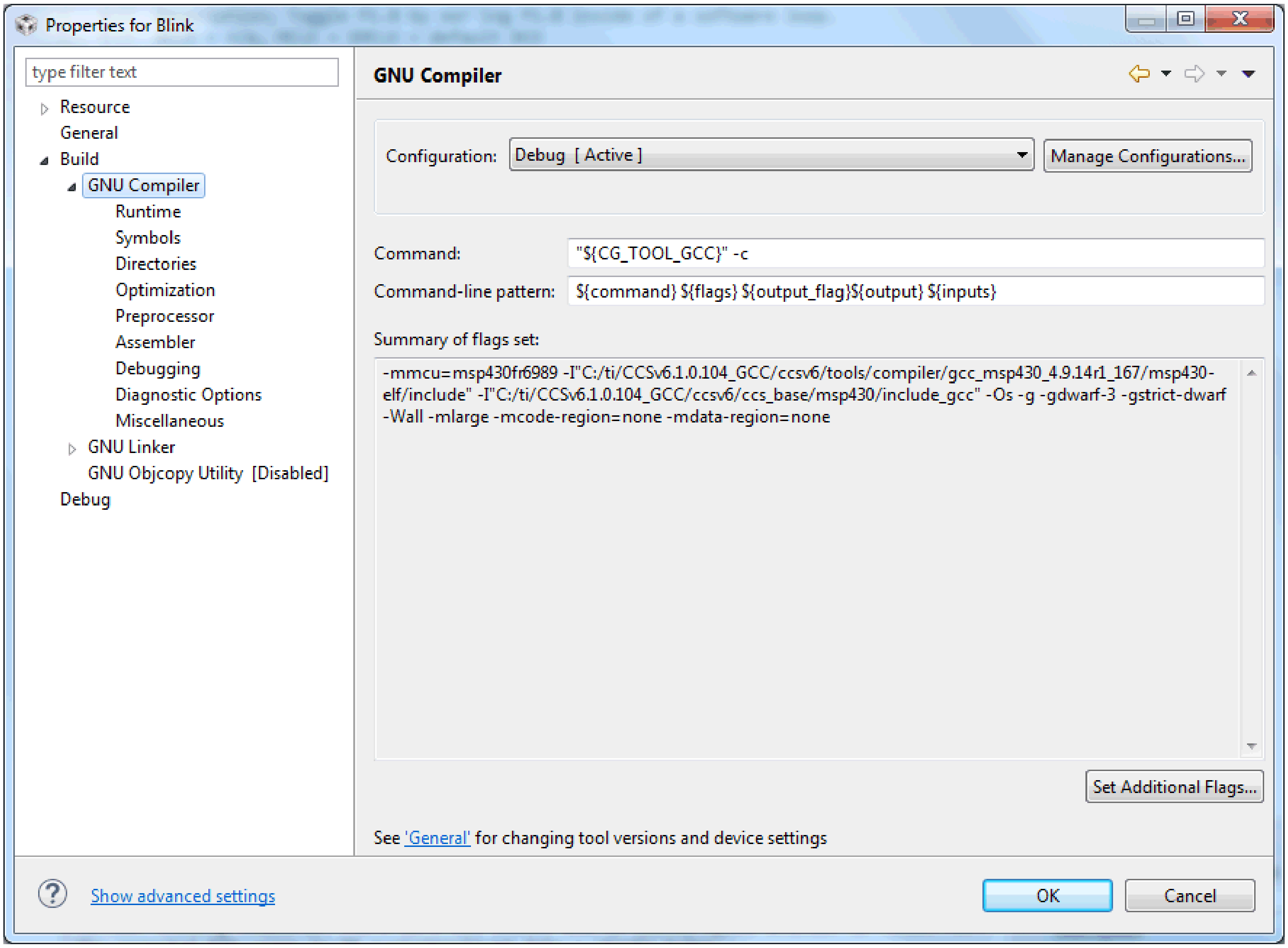
The following project settings are recommended or required:

- Specify the target device for debug session (**Project** → **Properties** → **General** → **Device** → **Variant**). The corresponding Linker Command File and Runtime Support Library are selected automatically.

- To more easily debug a C project, disable optimization (**Project** → **Properties** → **Build** → **GNU Compiler** → **Optimization** → **Optimization level**).
- Specify the search path for the C preprocessor (**Project** → **Properties** → **Build** → **GNU Compiler** → **Directories** → **Include Paths (-I)**).
- Specify the search path for any libraries being used (**Project** → **Properties** → **Build** → **GNU Linker** → **Libraries** → **Library search path (-L, --library-path)**).
- Specify the debugger interface (**Project** → **Properties** → **General** → **Device** → **Connection**). Select TI MSP430 USBx for the USB interface.
- Enable the erasure of the Main and Information memories before object code download (**Project** → **Properties** → **Debug** → **MSP430 Properties** → **Download Options** → **Erase Main and Information Memory**).
- To ensure proper stand-alone operation, select Hardware Breakpoints (**Project** → **Properties** → **Debug** → **MSP430 Properties**). If Software Breakpoints are enabled (**Project** → **Properties** → **Debug** → **Misc/Other Options** → **Allow software breakpoints to be used**), ensure proper termination of each debug session while the target is connected; otherwise, the target may not be operational stand-alone as the application on the device still contains the software breakpoint instructions.

3.3.1 GNU Compiler

Figure 8 shows the MSP430 GCC settings window.



Configuration: Debug [Active]

Command: "${CG_TOOL_GCC}" -c

Command-line pattern: ${command} ${flags} ${output_flag}${output} ${inputs}

Summary of flags set:

```
-mmcu=msp430fr6989 -I"C:/ti/CCSv6.1.0.104_GCC/ccsv6/tools/compiler/gcc_msp430_4.9.14r1_167/msp430-elf/include" -I"C:/ti/CCSv6.1.0.104_GCC/ccsv6/ccs_base/msp430/include_gcc" -Os -g -gdwarf-3 -gstrict-dwarf -Wall -mlarge -mcode-region=none -mdata-region=none
```

See 'General' for changing tool versions and device settings

Figure 8. MSP430 GCC Settings

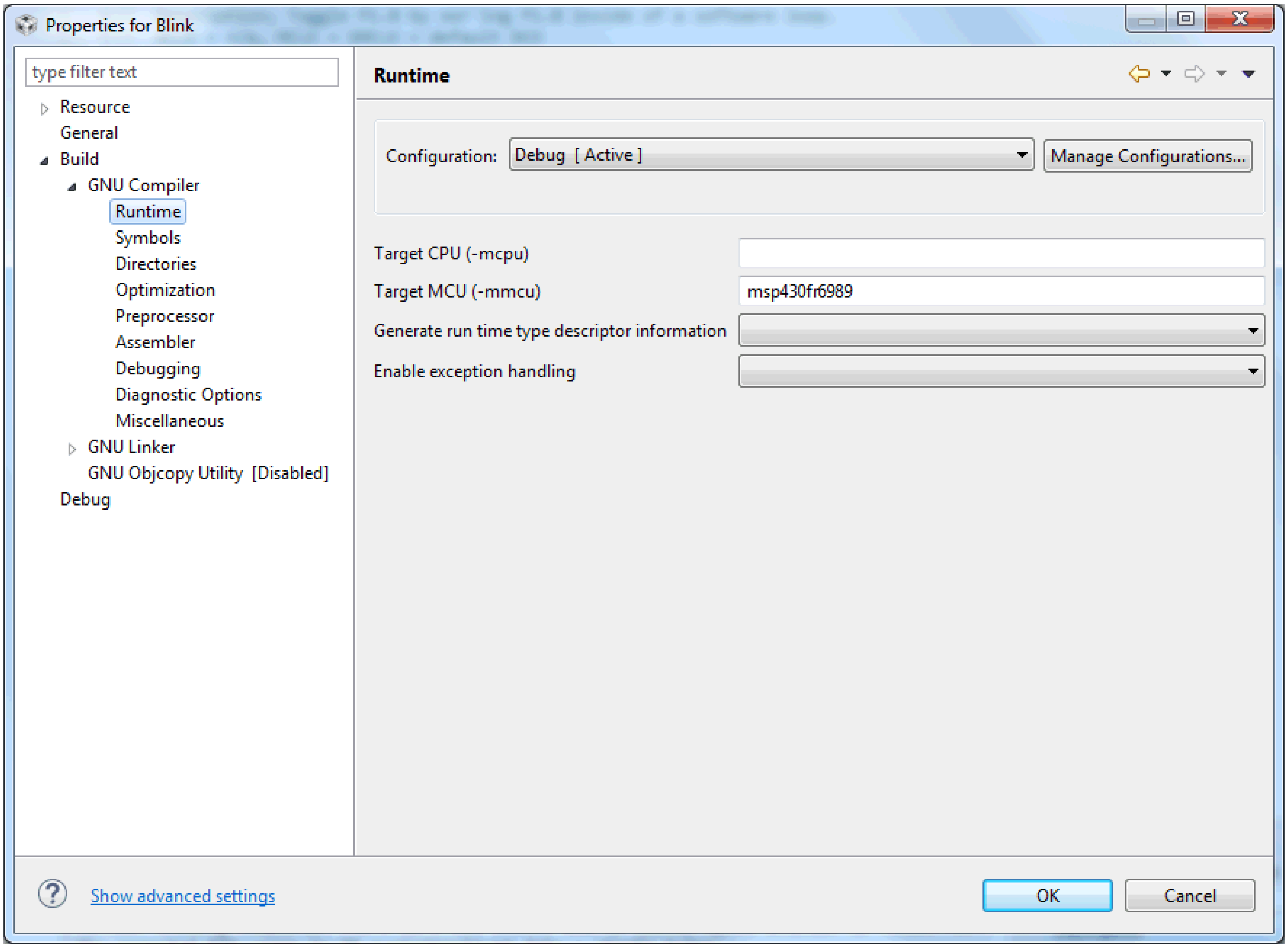
Table 2 describes the options that are available for MSP430 GCC Settings.

Table 2. MSP430 GCC Settings

Option	Description
Command	Compiler location
Command-line pattern	Command line parameters
Summary of flags set	Command line with which the compiler is called

3.3.2 GNU Compiler: Runtime

Figure 9 shows the MSP430 GCC Runtime settings window.



Configuration: Debug [Active]

Target CPU (-mcpu)

Target MCU (-mmcu) msp430fr6989

Generate run time type descriptor information

Enable exception handling

Figure 9. MSP430 GCC Settings: Runtime

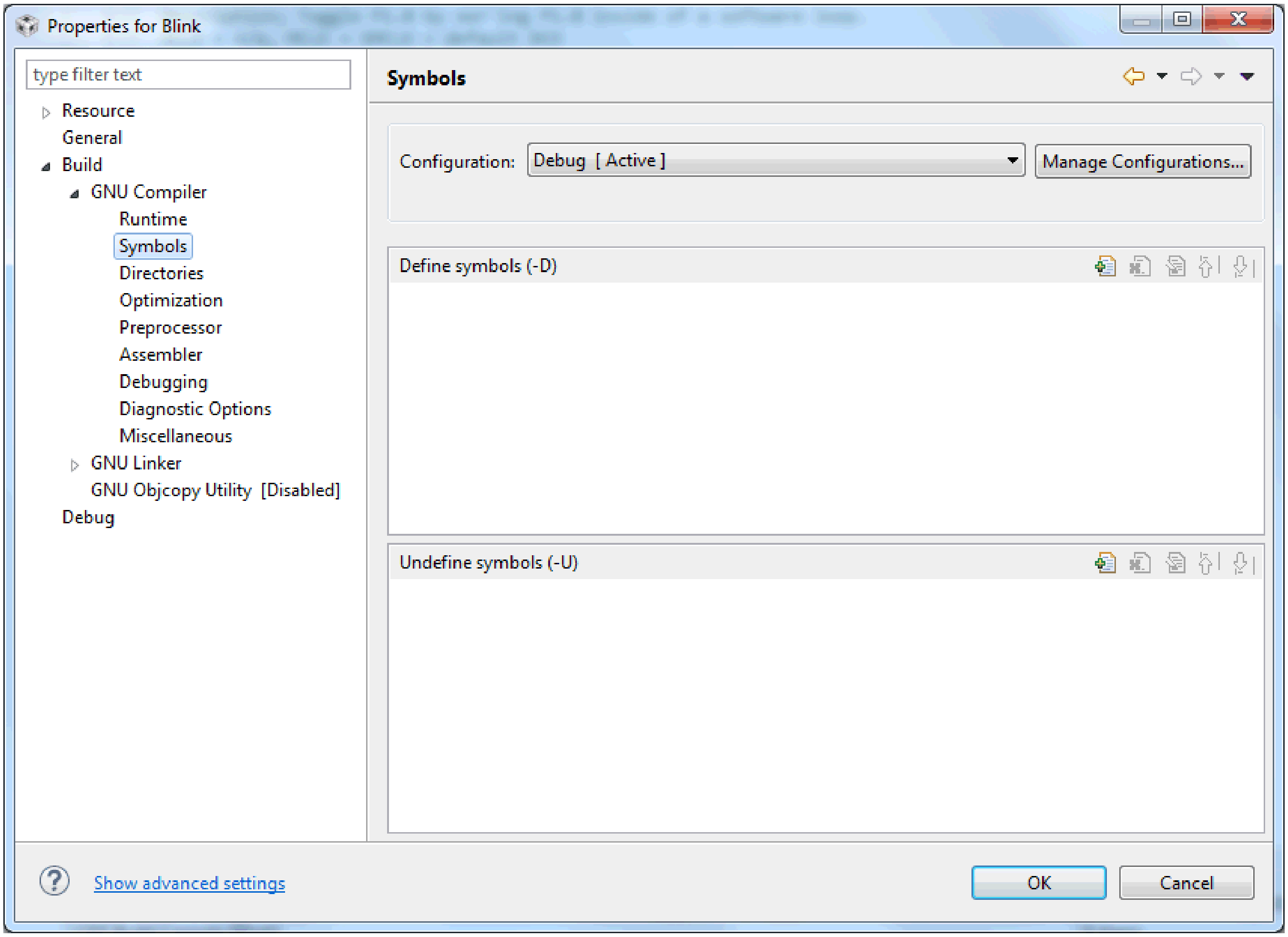
Table 3 describes the options that are available for MSP430 GCC Runtime settings.

Table 3. MSP430 GCC Settings: Runtime

Option	Description
Target CPU (-mcpu)	Specifies the ISA to use. Accepted values are msp430, msp430x, and msp430xv2. This option is deprecated. The '-mmcu=' option should be used to select the ISA.
Target MCU (-mmcu)	Select the MCU to target. This is used to create a C preprocessor symbol based on the MCU name, converted to upper case and prefix and postfixed with__. This in turn is used by the msp430.h header file to select an MCU-specific supplementary header file. The option also sets the ISA to use. If the MCU name is one that is known to only support the 430 ISA then that is selected, otherwise the 430X ISA is selected. A generic MCU name of msp430 can also be used to select the 430 ISA. Similarly the generic msp430x MCU name selects the 430X ISA. In addition, an MCU-specific linker script is added to the linker command line. The script's name is the name of the MCU with ".ld" appended. Thus, specifying '-mmcu=xxx' on the gcc command line defines the C preprocessor symbol __XXX__ and causes the linker to search for a script called 'xxx.ld'. This option is also passed to the assembler.
Generate run time type descriptor information	Enable or disable generation of information about every class with virtual functions for use by the C++ runtime type identification features. • On (-frtti) • Off (-fno-rtti)
Enable exception handling	Enable or disable exception handling. Generates extra code needed to propagate exceptions. • On (-fexceptions) • Off (-fno-exceptions)

3.3.3 GNU Compiler: Symbols

Figure 10 shows the MSP430 GCC Symbols settings window.



Configuration: Debug [Active]

Define symbols (-D)

Undefine symbols (-U)

Figure 10. MSP430 GCC Settings: Symbols

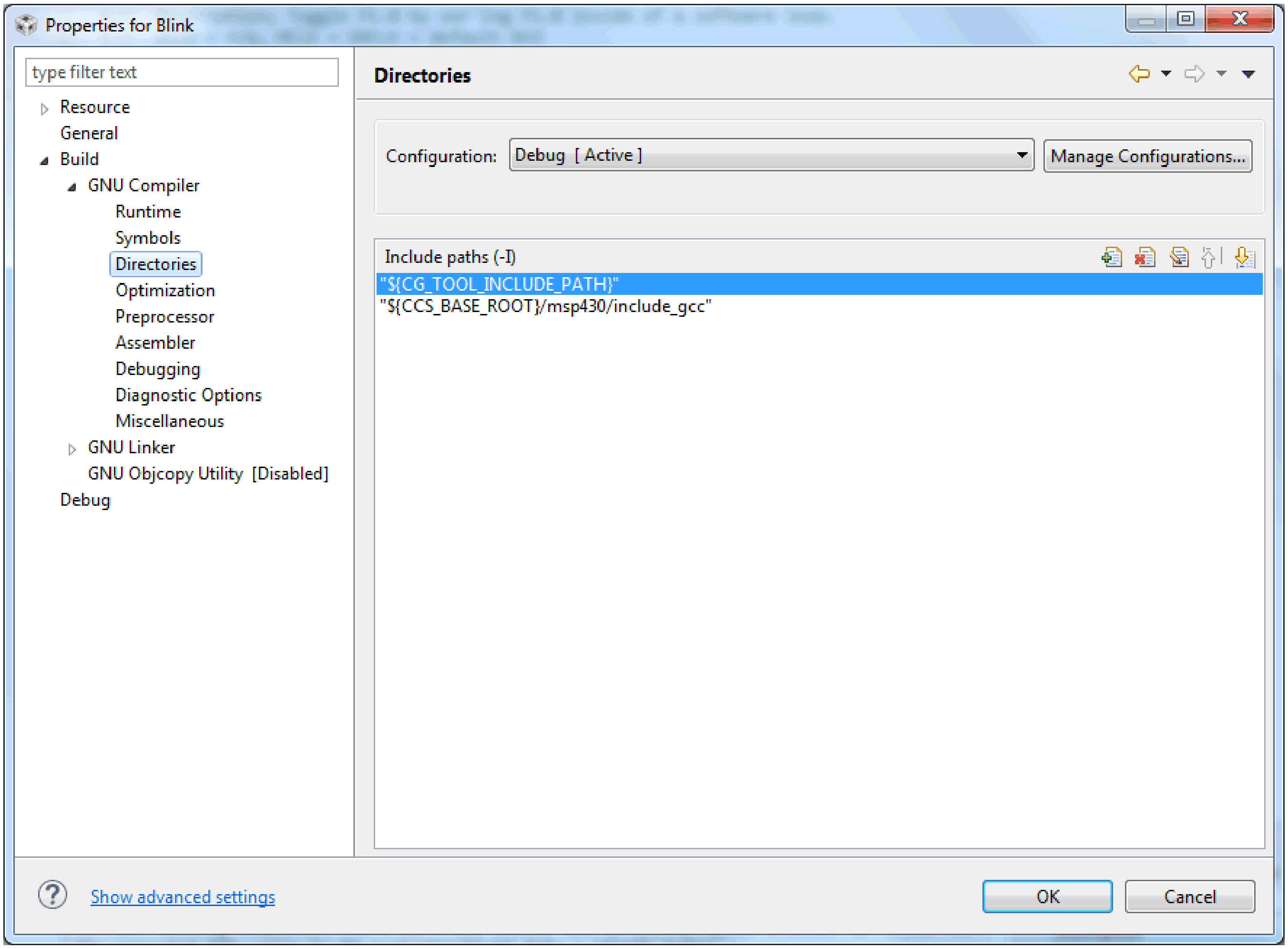
Table 4 describes the options that are available for MSP430 GCC Symbols settings.

Table 4. MSP430 GCC Settings: Symbols

Option	Description
Define symbols (-D)	-D name Predefine name as a macro, with definition 1. -D name=definition
Undefine symbols (-U)	Cancel any previous definition of name, either built in or provided with a -D option.

3.3.4 GNU Compiler: Directories

Figure 11 shows the MSP430 GCC Directories settings window.



Configuration: Debug [Active]

Include paths (-I)

```
"${CG_TOOL_INCLUDE_PATH}"
"${CCS_BASE_ROOT}/msp430/include_gcc"
```

Figure 11. MSP430 GCC Settings: Directories

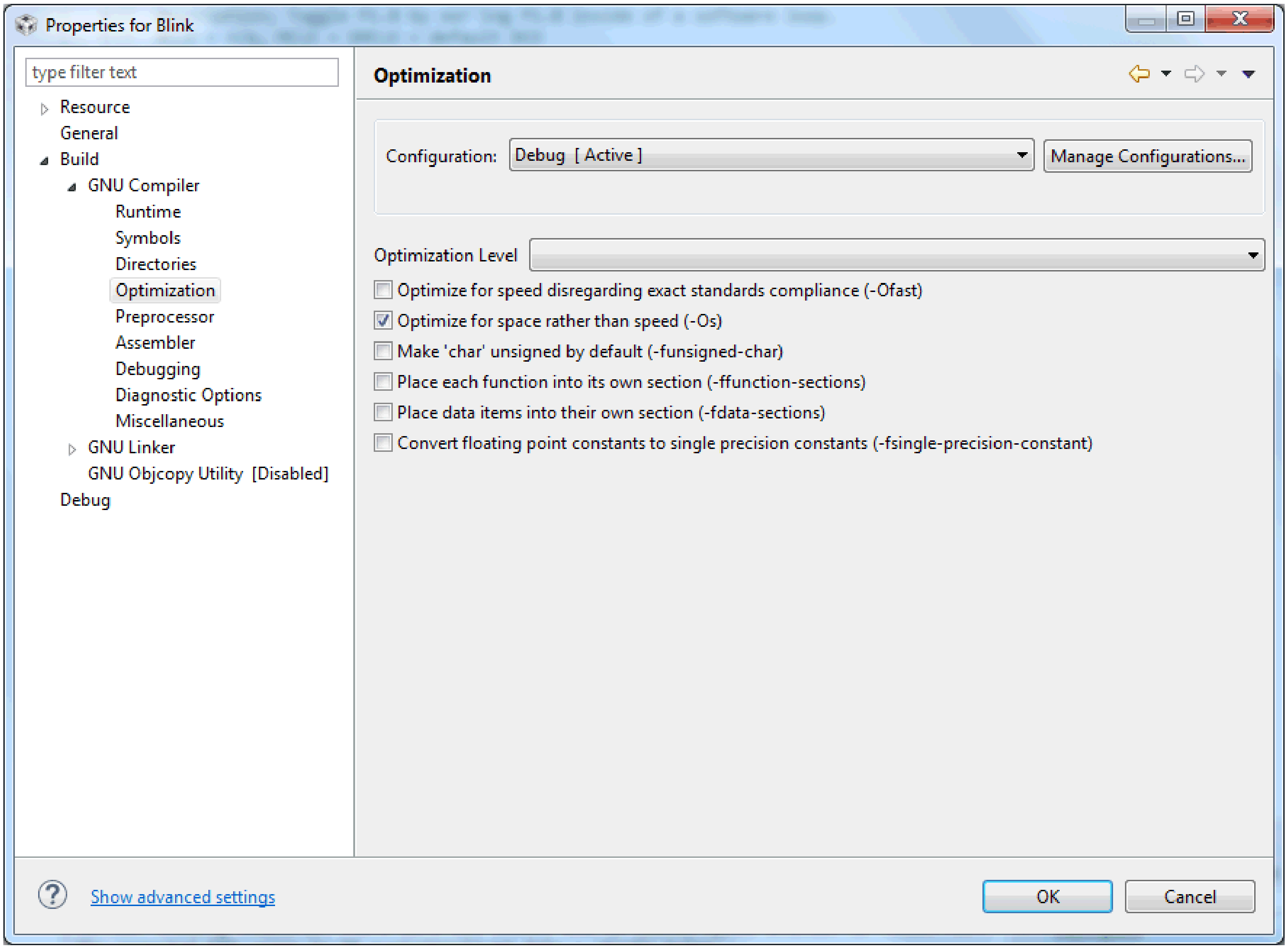
Table 5 describes the options that are available for MSP430 GCC Directories settings.

Table 5. MSP430 GCC Settings: Directories

Option	Description
Include paths (-I)	Add the directory to the list of directories to be searched for header files.

3.3.5 GNU Compiler: Optimization

Figure 12 shows the MSP430 GCC Optimization settings window.



Configuration: Debug [Active]

Optimization Level

□ Optimize for speed disregarding exact standards compliance (-Ofast)

☑ Optimize for space rather than speed (-Os)

□ Make 'char' unsigned by default (-funsigned-char)

□ Place each function into its own section (-ffunction-sections)

□ Place data items into their own section (-fdata-sections)

□ Convert floating point constants to single precision constants (-fsingle-precision-constant)

Figure 12. MSP430 GCC Settings: Optimization

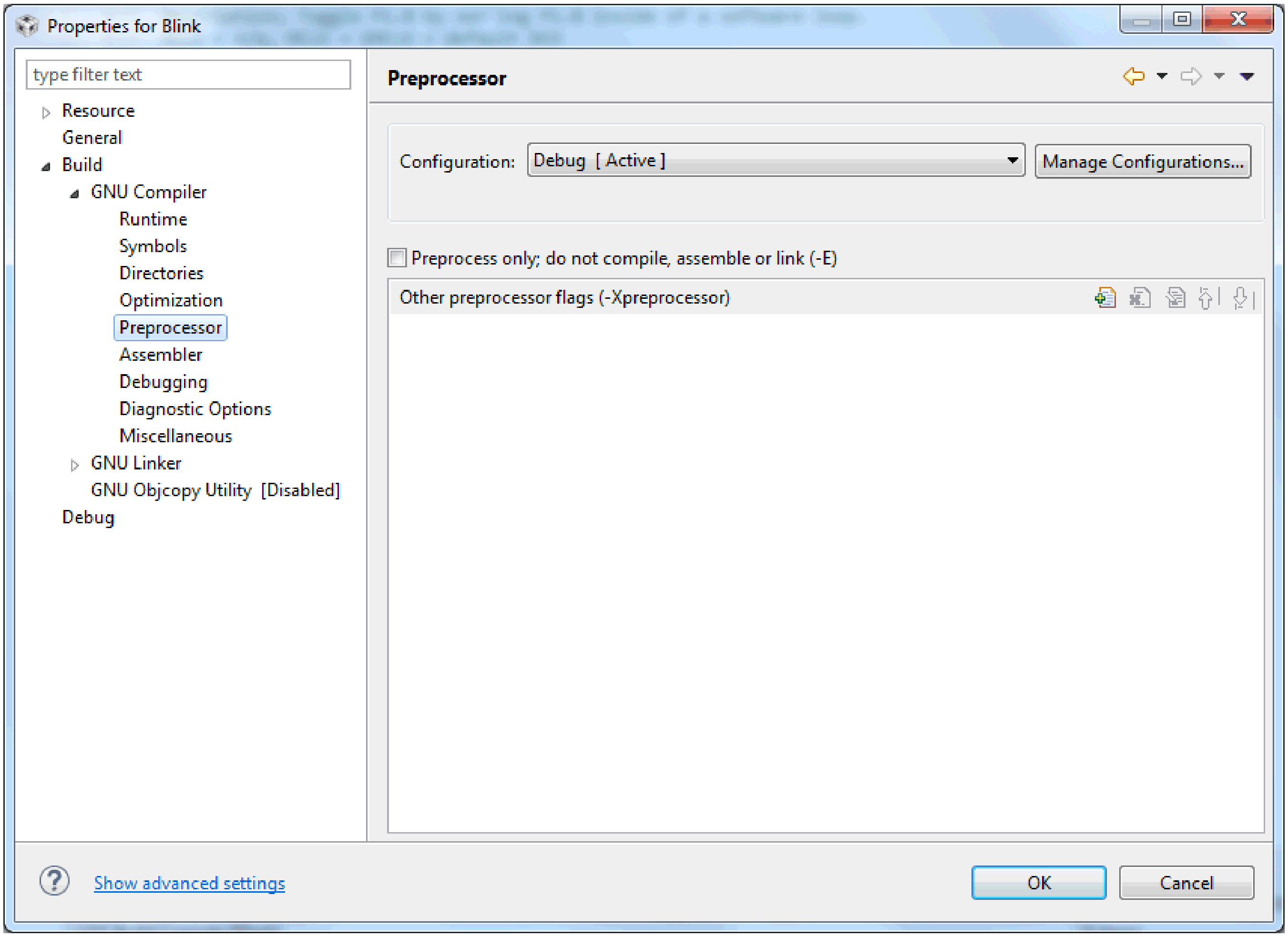
Table 6 describes the options that are available for MSP430 GCC Optimization settings.

Table 6. MSP430 GCC Settings: Optimization

Option	Description
Optimization Level	Specifies the optimizations that the compiler applies to the generated object code. The options available are: • None (O0): Disable optimizations. This setting is equivalent to specifying the -O0 command-line option. The compiler generates unoptimized linear assembly language code. • Optimize (O1): The compiler performs all targets independent (that is, nonparallelized) optimizations, such as function inlining. This setting is equivalent to specifying the -O1 command-line option. The compiler omits all target-specific optimizations and generates linear assembly language code. • Optimize more (O2): The compiler performs all optimizations (both target-independent and target-specific). This setting is equivalent to specifying the -O2 command-line option. The compiler outputs optimized nonlinear parallelized assembly language code. • Optimize most (O3): The compiler performs all the level 2 optimizations, then the low-level optimizer performs global-algorithm register allocation. This setting is equivalent to specifying the -O3 command-line option. At this optimization level, the compiler generates code that is usually faster than the code generated from level 2 optimizations.
Optimize for speed disregarding exact standards compliance (-Ofast)	Disregard strict standards compliance. -Ofast enables all -O3 optimizations. It also enables optimizations that are not valid for all standard-compliant programs. It turns on -ffast-math and the Fortran-specific -fno-protect-parens and -fstack-arrays.
Optimize for space rather than speed (-Os)	Optimize for size (-Os) –Os enables all -O2 optimizations that do not typically increase code size. It also performs further optimizations designed to reduce code size. This setting is equivalent to specifying the -Os command-line option.
Make 'char' unsigned by default (-funsigned-char)	Enable this option to ensure that the char is signed.
Place each function into its own section (-ffunction-sections)	Enable this option to use function sections and is equivalent to -ffunction-sections.
Place data items into their own section (-fdata-sections)	Enable this option to use short data sections and is equivalent to -ffunction-sections.
Convert floating point constants to single precision constants (-fsingle-precision-constant)	Treat floating-point constants as single precision instead of implicitly converting them to double-precision constants.

3.3.6 GNU Compiler: Preprocessor

Figure 13 shows the MSP430 GCC Preprocessor settings window.



Configuration: Debug [Active]

□ Preprocess only; do not compile, assemble or link (-E)

Other preprocessor flags (-Xpreprocessor)

Figure 13. MSP430 GCC Settings: Preprocessor

Table 7 describes the options that are available for MSP430 GCC Preprocessor settings.

Table 7. MSP430 GCC Settings: Preprocessor

Option	Description
Preprocess only; do not compile, assemble or link (-E)	Enable this option to preprocess only without compiling or assembling or linking.
Other preprocessor flags (-Xpreprocessor)	Use this to supply system-specific preprocessor options that GCC does not recognize. To pass an option that takes an argument, use -Xpreprocessor twice, once for the option and once for the argument.

3.3.7 GNU Compiler: Assembler

Figure 14 shows the MSP430 GCC Assembler settings window.

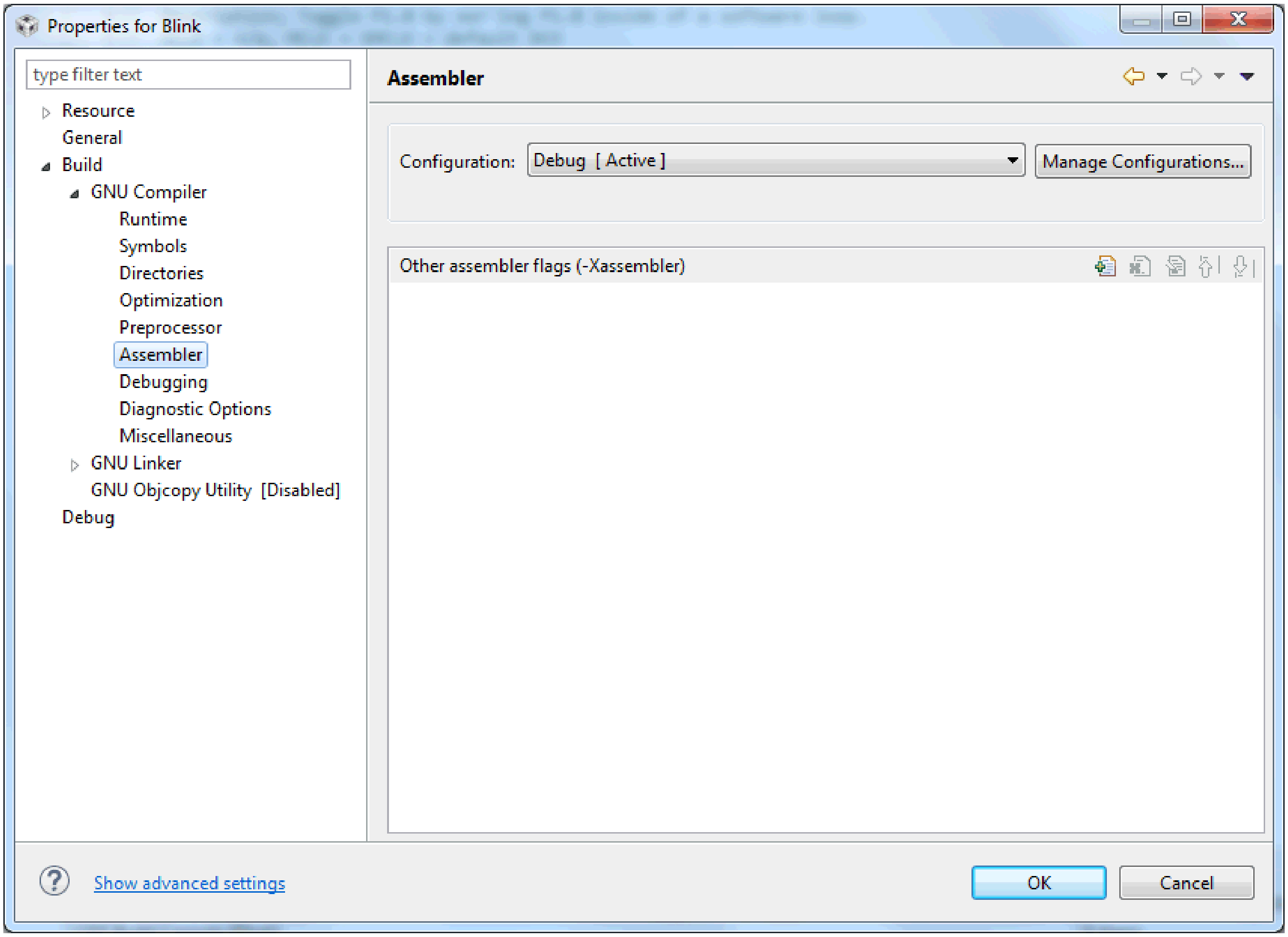


Figure 14. MSP430 GCC Settings: Assembler

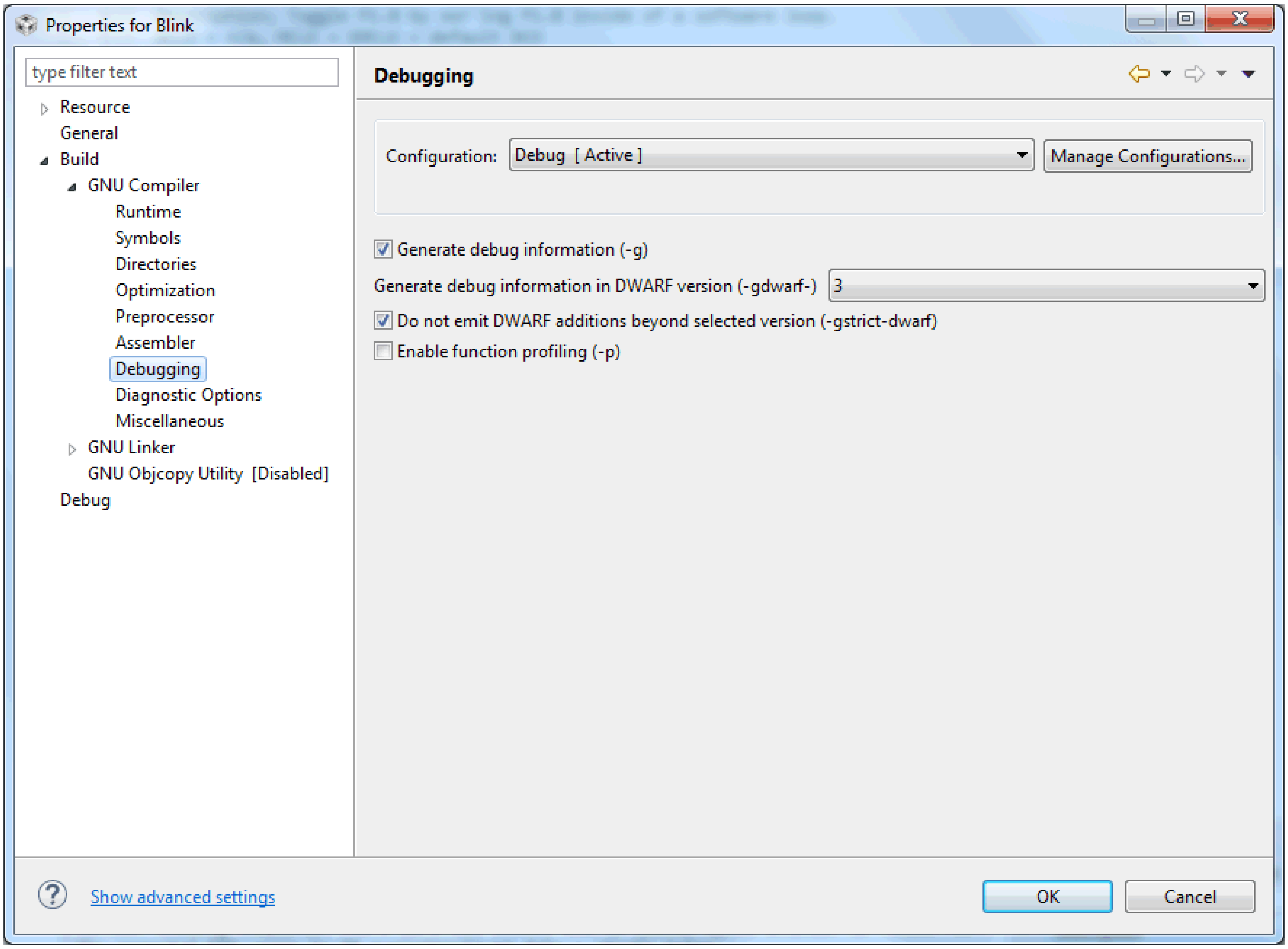
Table 8 describes the options that are available for MSP430 GCC Assembler settings.

Table 8. MSP430 GCC Settings: Assembler

Option	Description
Other assembler flags (-Xassembler)	Specifies individual flag based on the user requirements.

3.3.8 GNU Compiler: Debugging

Figure 15 shows the MSP430 GCC Debugging settings window.



Debugging

Configuration: Debug [Active]

☑ Generate debug information (-g)

Generate debug information in DWARF version (-gdwarf-) 3

☑ Do not emit DWARF additions beyond selected version (-gstrict-dwarf)

□ Enable function profiling (-p)

Figure 15. MSP430 GCC Settings: Debugging

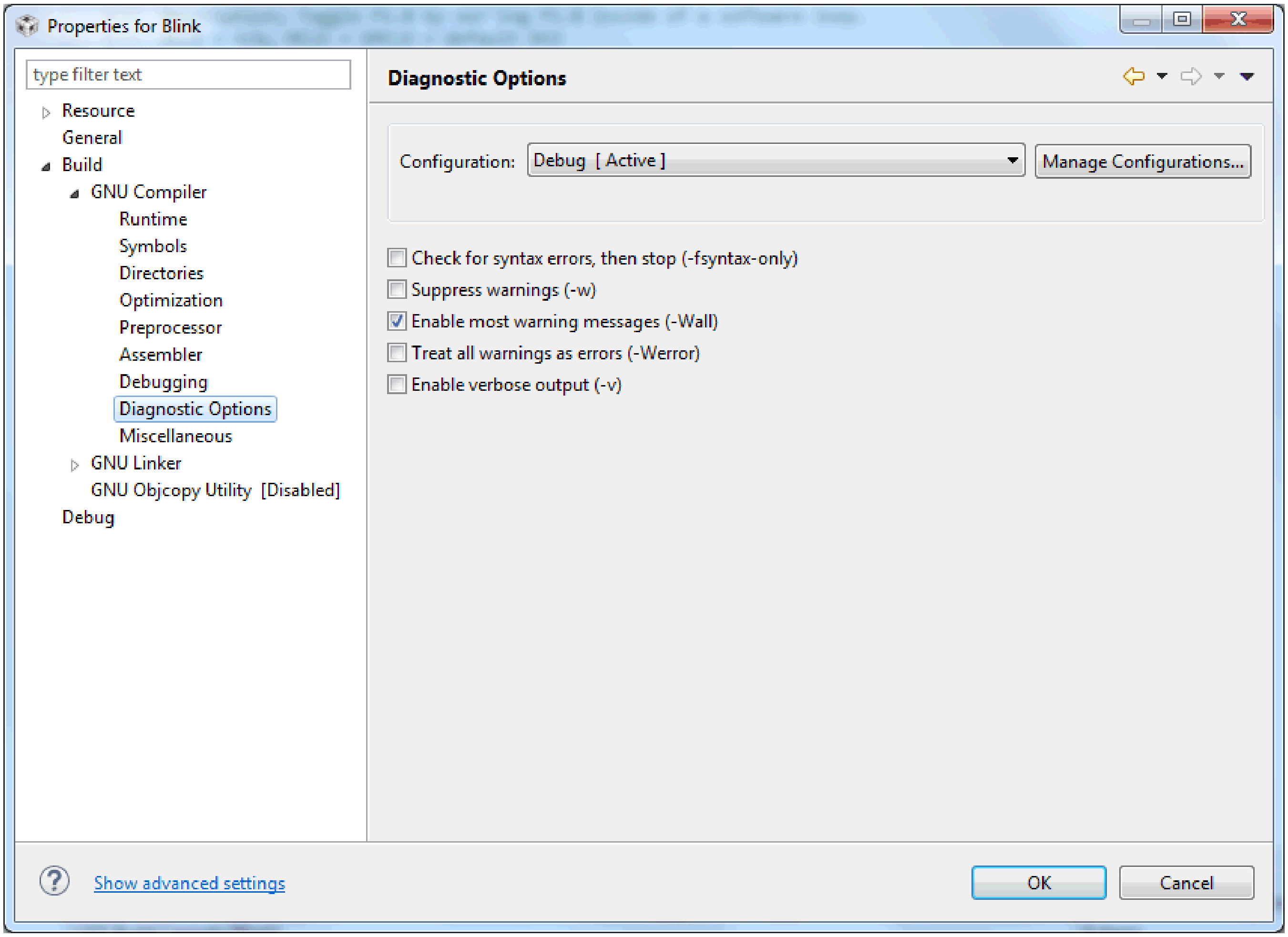
Table 9 describes the options that are available for MSP430 GCC Debugging settings.

Table 9. MSP430 GCC Settings: Debugging

Option	Description
Generate debug information (-g)	Produce debugging information. GDB can work with this debugging information.
Generate debug information in DWARF version (-gdwarf-)	Produce debugging information in DWARF format (if that is supported). The value of version may be 2, 3 or 4; the default version for most targets is 4.
Do not emit DWARF additions beyond selected version (-gstrict-dwarf)	Disallow using extensions of later DWARF standard version than selected with -gdwarf-version. On most targets using nonconflicting DWARF extensions from later standard versions is allowed.
Enable function profiling (-p)	Generate extra code to write profile information suitable for the analysis program. This option is required when compiling source files for which data is needed, and it is also required when linking.

3.3.9 GNU Compiler: Diagnostic Options

Figure 16 shows the MSP430 GCC Diagnostic Options settings window.



Diagnostic Options

Configuration: Debug [Active]

□ Check for syntax errors, then stop (-fsyntax-only)

□ Suppress warnings (-w)

☑ Enable most warning messages (-Wall)

□ Treat all warnings as errors (-Werror)

□ Enable verbose output (-v)

Figure 16. MSP430 GCC Settings: Diagnostic Options

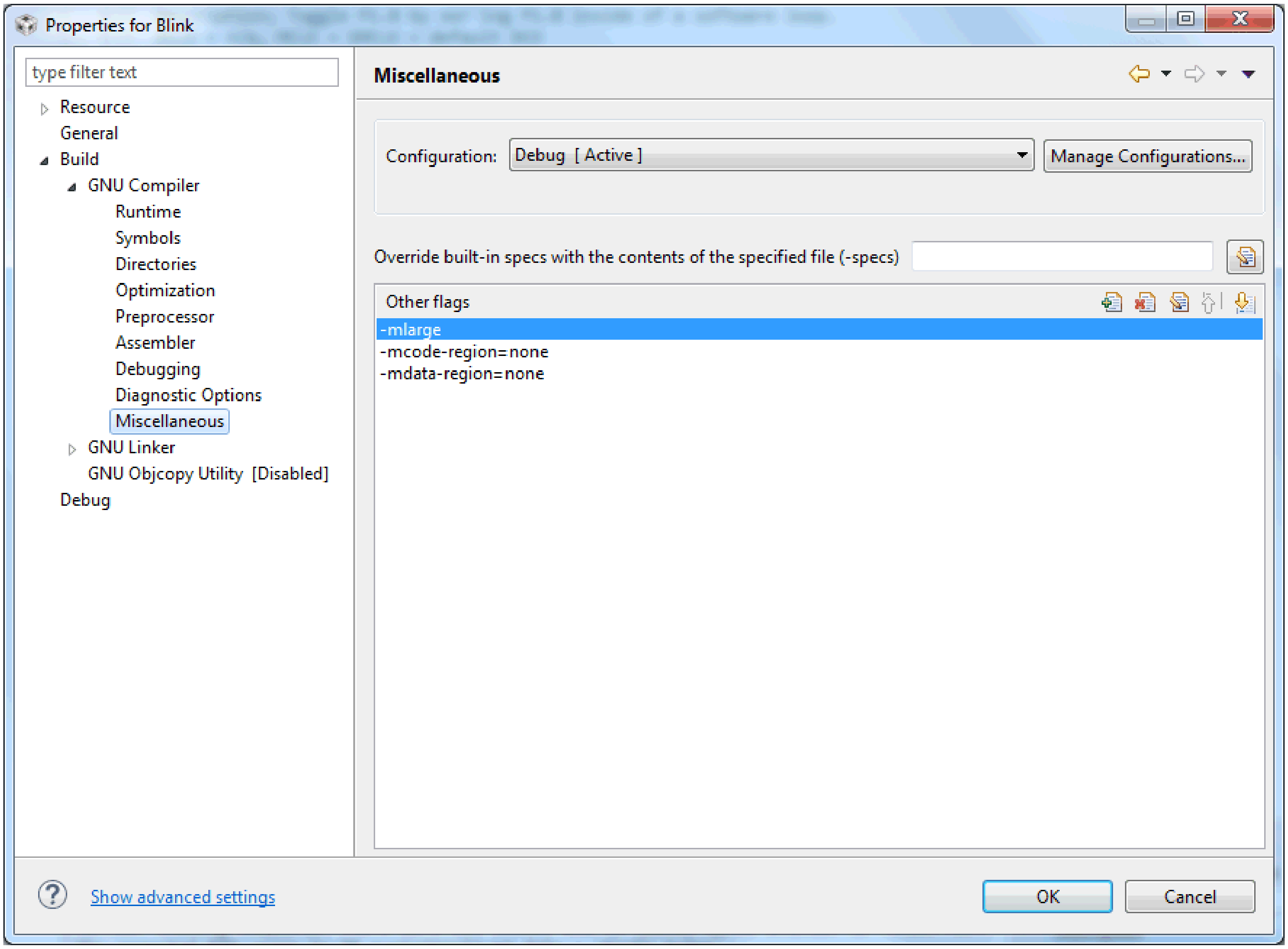
Table 10 describes the options that are available for MSP430 GCC Diagnostic Options settings.

Table 10. MSP430 GCC Settings: Diagnostic Options

Option	Description
Check for syntax errors, then stop (-fsyntax-only)	Enable this option to check the syntax of commands and throw a syntax error.
Suppress warnings (-w)	Enable this option to enable all the warnings about constructions that some users consider questionable and that are easy to avoid (or modify to prevent the warning), even in conjunction with macros.
Enable most warning messages (-Wall)	Enable this option to enable all the warnings about constructions that some users consider questionable, and that are easy to avoid (or modify to prevent the warning), even in conjunction with macros.
Treat all warnings as errors (-Werror)	Enable this option to make all warnings into hard errors. Source code that triggers warnings is rejected.
Enable verbose output (-v)	Enable this option for the IDE to show each command line that it passes to the shell, along with all progress, error, warning, and informational messages that the tools emits. This setting is equivalent to specifying the –v command-line option. By default, this checkbox is clear. The IDE displays only error messages that the compiler emits. The IDE suppresses warning and informational messages.

3.3.10 GNU Compiler: Miscellaneous

Figure 17 shows the MSP430 GCC Miscellaneous settings window.



Configuration: Debug [Active]

Override built-in specs with the contents of the specified file (-specs)

Other flags
-mlarge
-mcode-region=none
-mdata-region=none

Figure 17. MSP430 GCC Settings: Miscellaneous

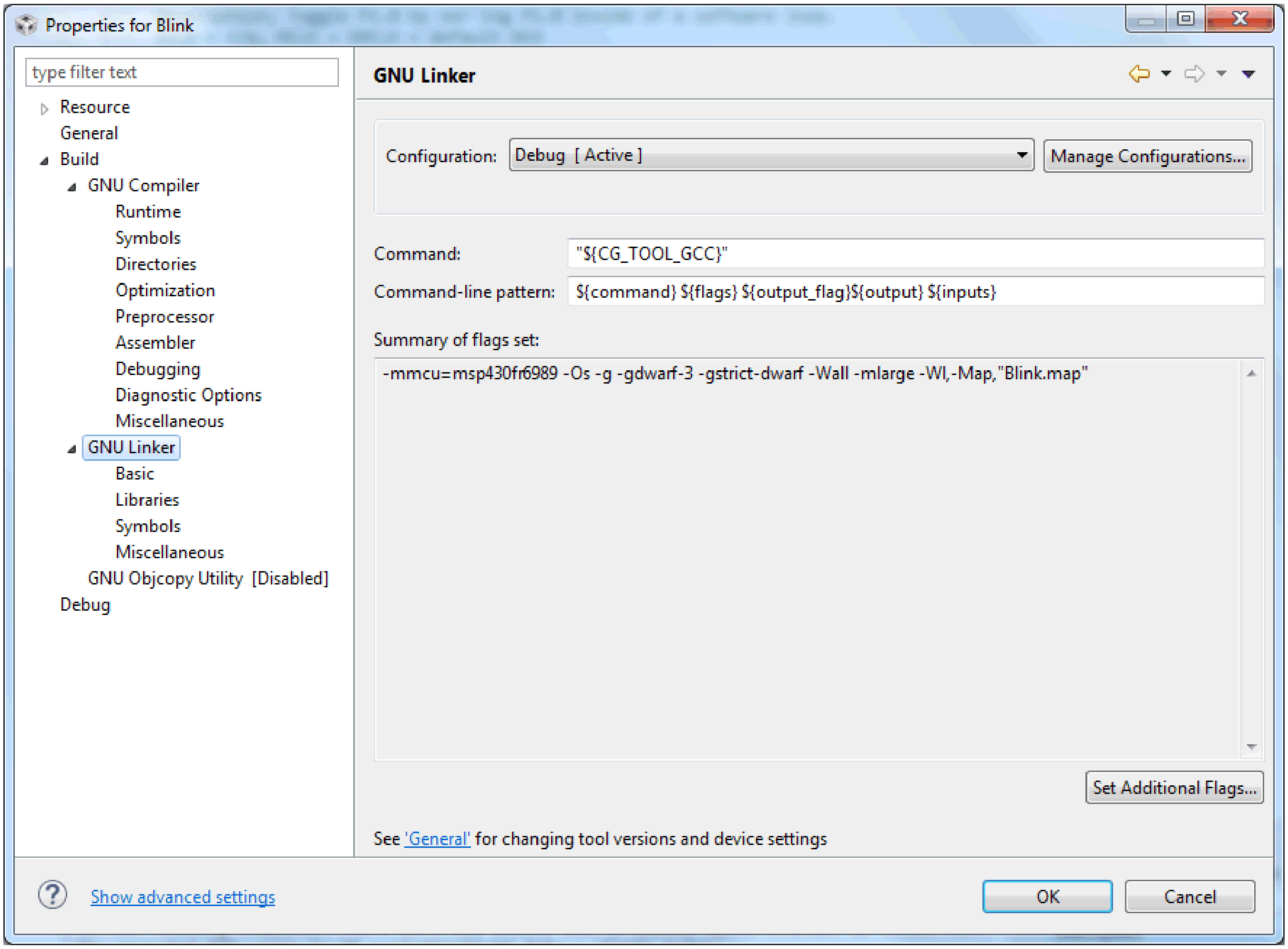
Table 11 describes the options that are available for MSP430 GCC Miscellaneous settings.

Table 11. MSP430 GCC Settings: Miscellaneous

Option	Description
Override built-in specs with the contents of the specified file (-specs)	The spec strings built into GCC can be overridden by using the -specs= command-line switch to specify a spec file.
Other flags	-mlarge Use large-model addressing (20-bit pointers, 32-bit size_t). -mcode-region=none -mdata-region=none The MSP430 compiler has the ability to automatically distribute code and data between low memory (addresses below 64K) and high memory. This only applies to parts that actually have both memory regions and only if the linker script for the part has been specifically set up to support this feature.

3.3.11 GNU Linker

Figure 18 shows the MSP430 GCC Linker settings window.



Configuration: Debug [Active]

Command: "${CG_TOOL_GCC}"

Command-line pattern: ${command} ${flags} ${output_flag}${output} ${inputs}

Summary of flags set:

```
-mmcu=msp430fr6989 -Os -g -gdwarf-3 -gstrict-dwarf -Wall -mlarge -Wl,-Map,"Blink.map"
```

See 'General' for changing tool versions and device settings

Figure 18. MSP430 GCC Linker Settings

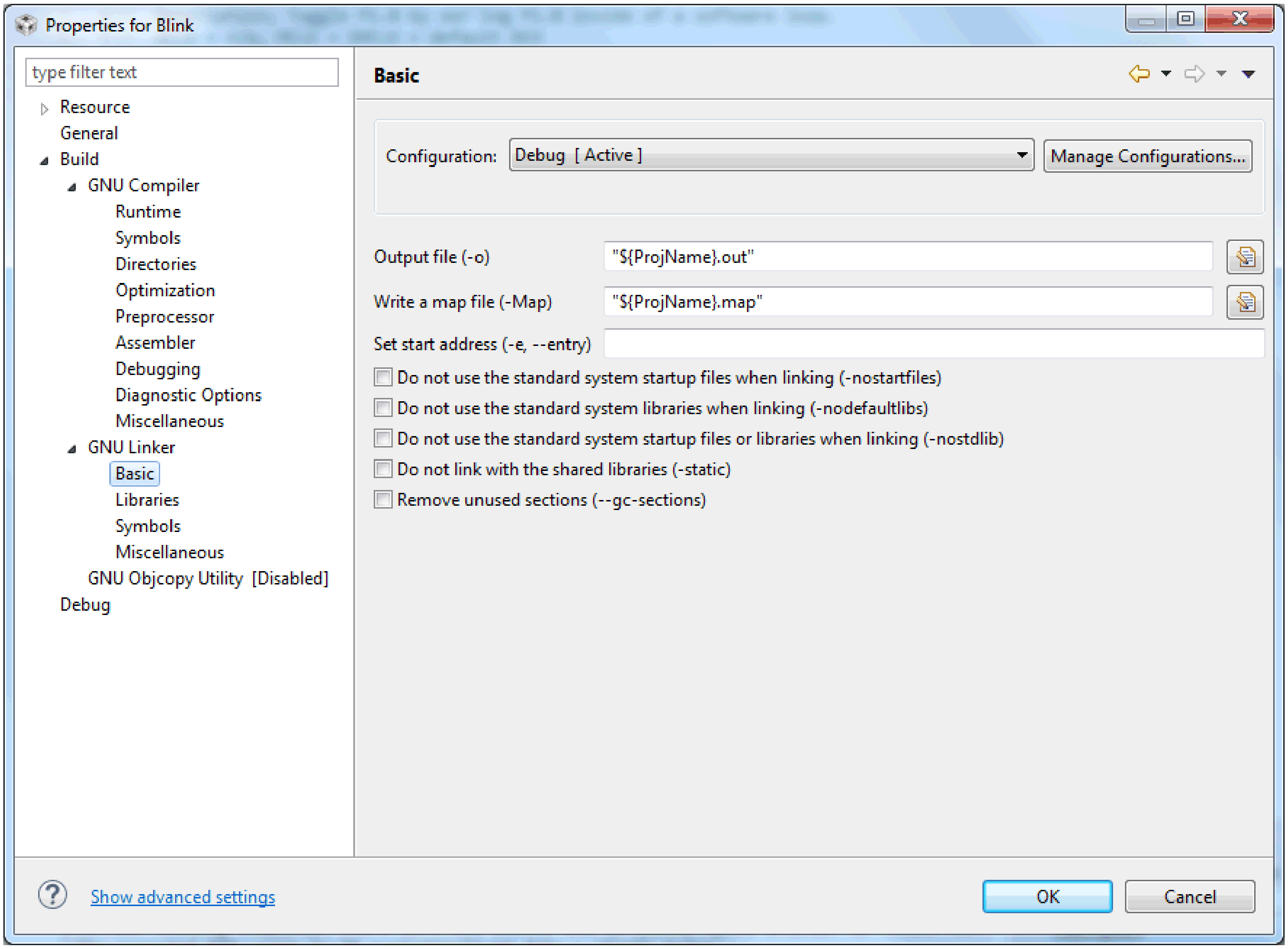
Table 12 describes the options that are available for MSP430 GCC Linker settings.

Table 12. MSP430 GCC Linker Settings

Option	Description
Command	Linker location
Command-line pattern	Command line parameters
Summary of flags set	Command line with which the compiler is called

3.3.12 GNU Linker: Basic

Figure 19 shows the MSP430 GCC Linker Basic settings window.



Configuration: Debug [Active]

Output file (-o) "${ProjName}.out"

Write a map file (-Map) "${ProjName}.map"

Set start address (-e, --entry)

☐ Do not use the standard system startup files when linking (-nostartfiles)

☐ Do not use the standard system libraries when linking (-nodefaultlibs)

☐ Do not use the standard system startup files or libraries when linking (-nostdlib)

☐ Do not link with the shared libraries (-static)

☐ Remove unused sections (--gc-sections)

Figure 19. MSP430 GCC Linker Basic Settings

Table 13 describes the options that are available for MSP430 GCC Linker Basic settings.

Table 13. MSP430 GCC Linker Basic Settings

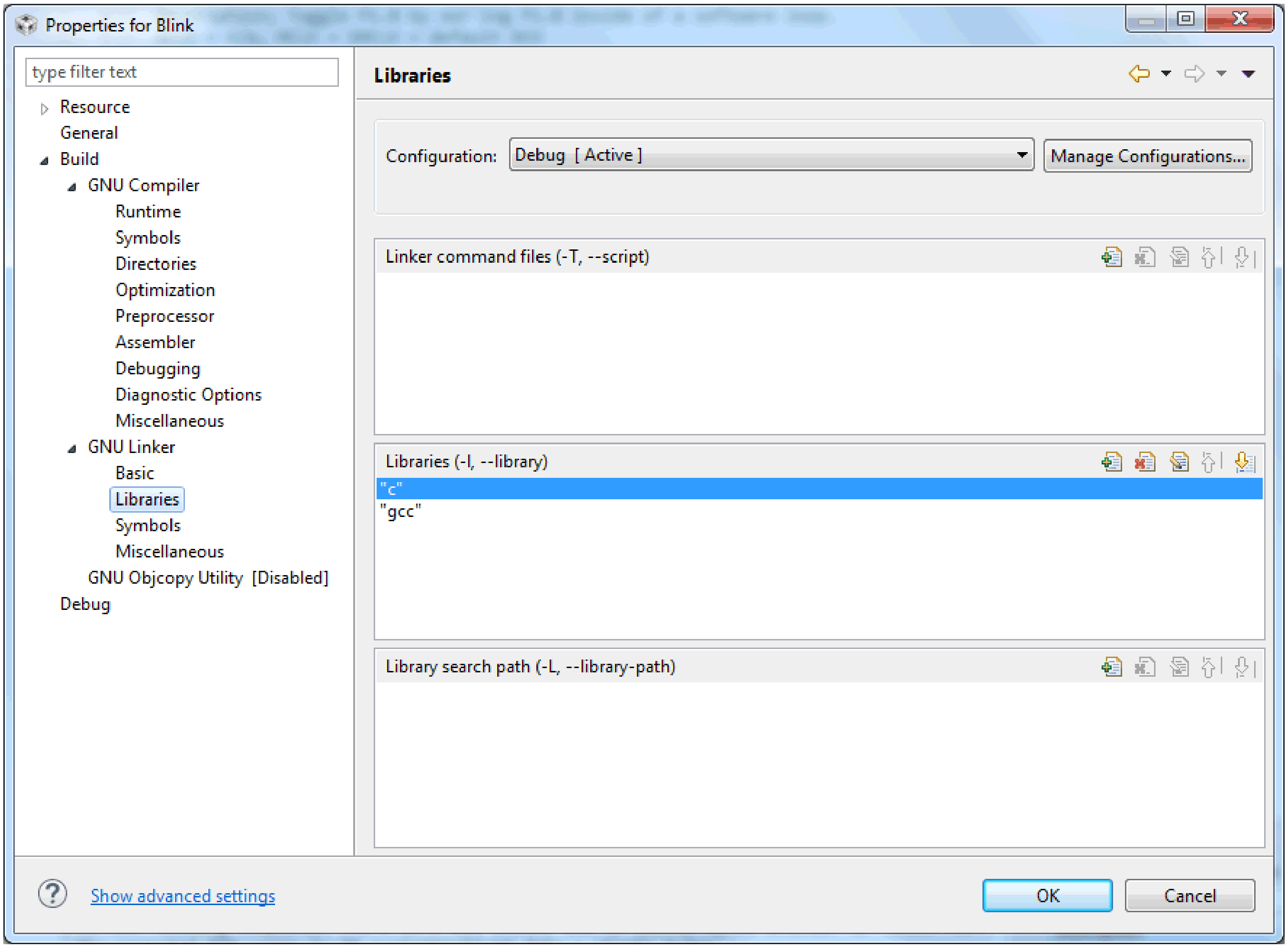
Option	Description
Output file (-o)	-o output Use output as the name for the program produced by ld; if this option is not specified, the name 'a.out' is used by default. The script command OUTPUT can also specify the output file name.
Write a map file (-Map)	Print to the file mapfile a link map, which contains diagnostic information about where symbols are mapped by ID and information on global common storage allocation.
Set start address (-e, --entry)	Use entry as the explicit symbol for beginning execution of the program, rather than the default entry point.
Do not use the standard system startup files when linking (-nostartfiles)	Do not use the standard system startup files when linking. The standard system libraries are used unless -nostdlib or -nodefaultlibs is used.

Table 13. MSP430 GCC Linker Basic Settings (continued)

Option	Description
Do not use the standard system libraries when linking (-nodefaultlibs)	Do not use the standard system libraries when linking. Only the specified libraries are passed to the linker, and options specifying linkage of the system libraries, such as -static-libgcc or -shared-libgcc, are ignored. The standard startup files are used unless -nostartfiles is used. The compiler may generate calls to memcmp, memset, memcpy, and memmove. These entries are usually resolved by entries in libc. These entry points should be supplied through some other mechanism when this option is specified.
Do not use the standard system startup files or libraries when linking (-nostdlib)	Do not use the standard system startup files or libraries when linking.
Do not link with the shared libraries (-static)	On systems that support dynamic linking, this prevents linking with the shared libraries. On other systems, this option has no effect.
Remove unused sections (--gc-sections)	Enable garbage collection of unused input sections. It is ignored on targets that do not support this option.

3.3.13 GNU Linker: Libraries

Figure 20 shows the MSP430 GCC Linker Libraries settings window.



Configuration: Debug [Active]

Linker command files (-T, --script)

Libraries (-l, --library)

"c"
"gcc"

Library search path (-L, --library-path)

Figure 20. MSP430 GCC Linker Libraries Settings

Table 14 describes the options that are available for MSP430 GCC Linker Libraries settings.

Table 14. MSP430 GCC Linker Libraries Settings

Option	Description
Linker command files (-T, --script)	-T commandfile Read link commands from the file command file.
Libraries (-l, --library)	-l library Search the library named library when linking.
Library search path (-L, --library-path)	-L searchdir Add path searchdir to the list of paths that ld will search for archive libraries and ld control scripts.

3.3.14 GNU Linker: Symbols

Figure 21 shows the MSP430 GCC Linker Symbols settings window.

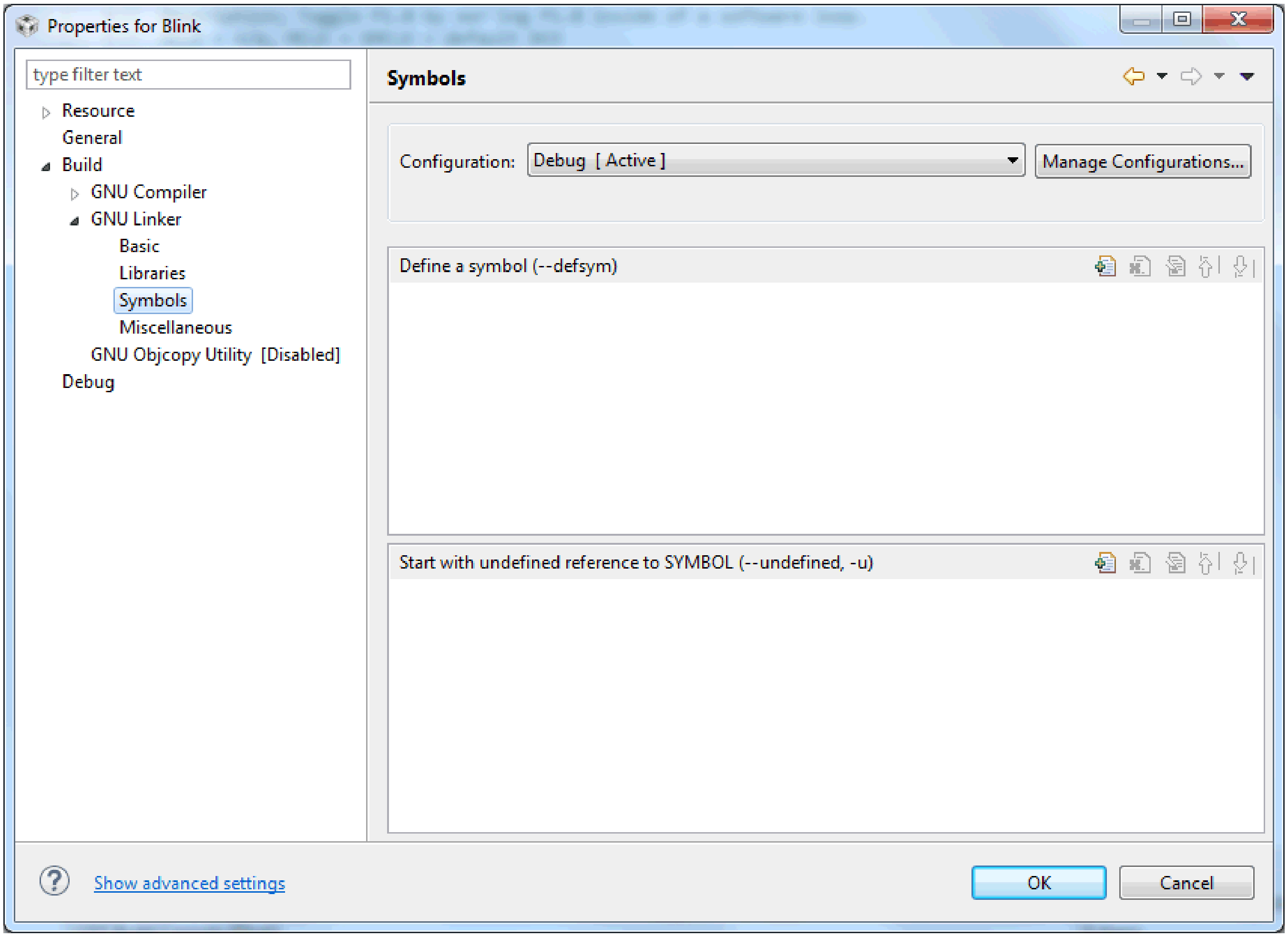


Figure 21. MSP430 GCC Linker Symbols Settings

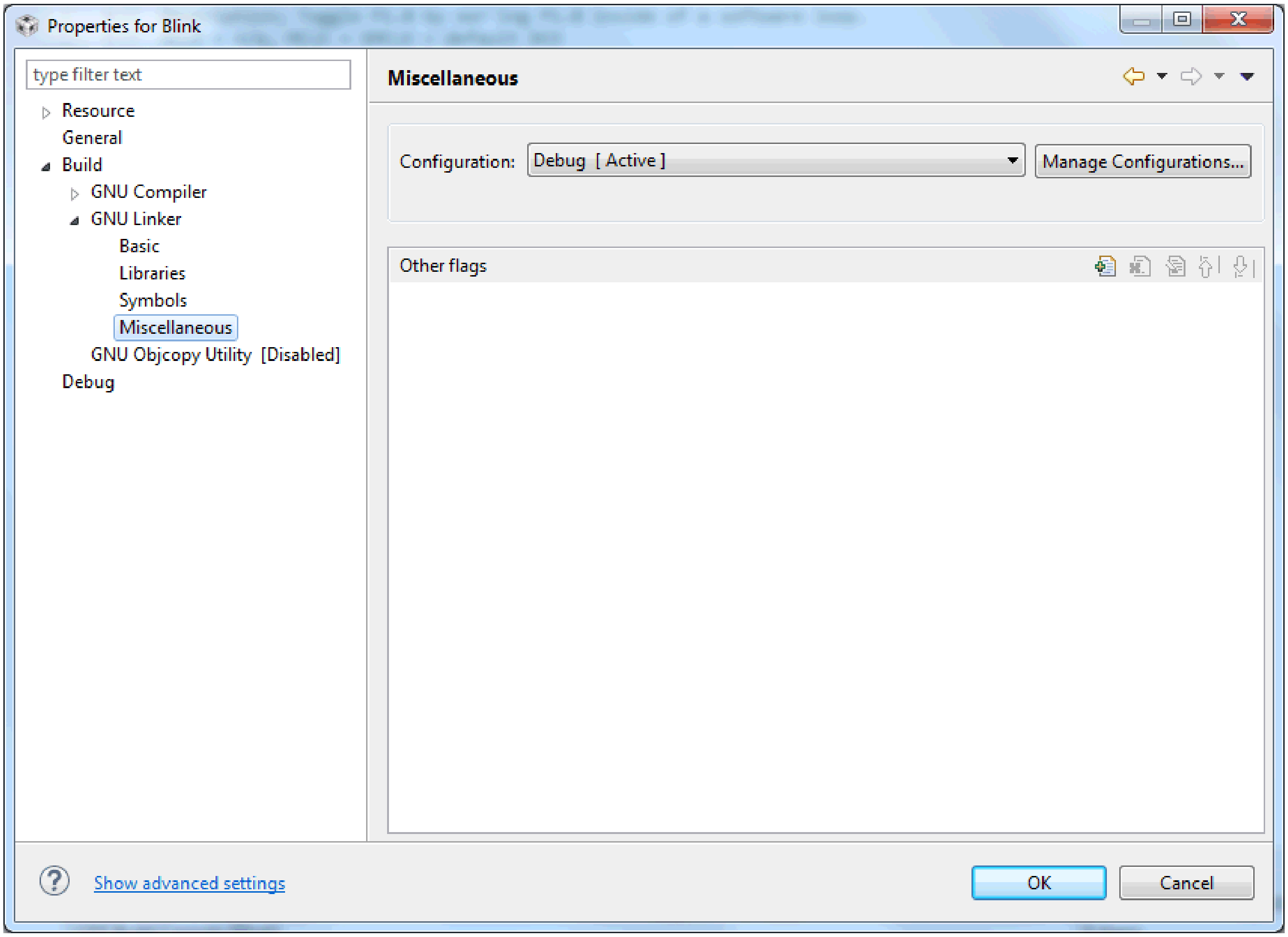
Table 15 describes the options that are available for MSP430 GCC Linker Symbols settings.

Table 15. MSP430 GCC Linker Symbols Settings

Option	Description
Define a symbol (--defsym)	-defsym symbol=expressionCreate a global symbol in the output file, containing the absolute address given by expression.
Start with undefined reference to SYMBOL (--undefined, -u)	Force symbol to be entered in the output file as an undefined symbol

3.3.15 GNU Linker: Miscellaneous

Figure 22 shows the MSP430 GCC Linker Miscellaneous settings window.



Configuration: Debug [Active]

Figure 22. MSP430 GCC Linker Miscellaneous Settings

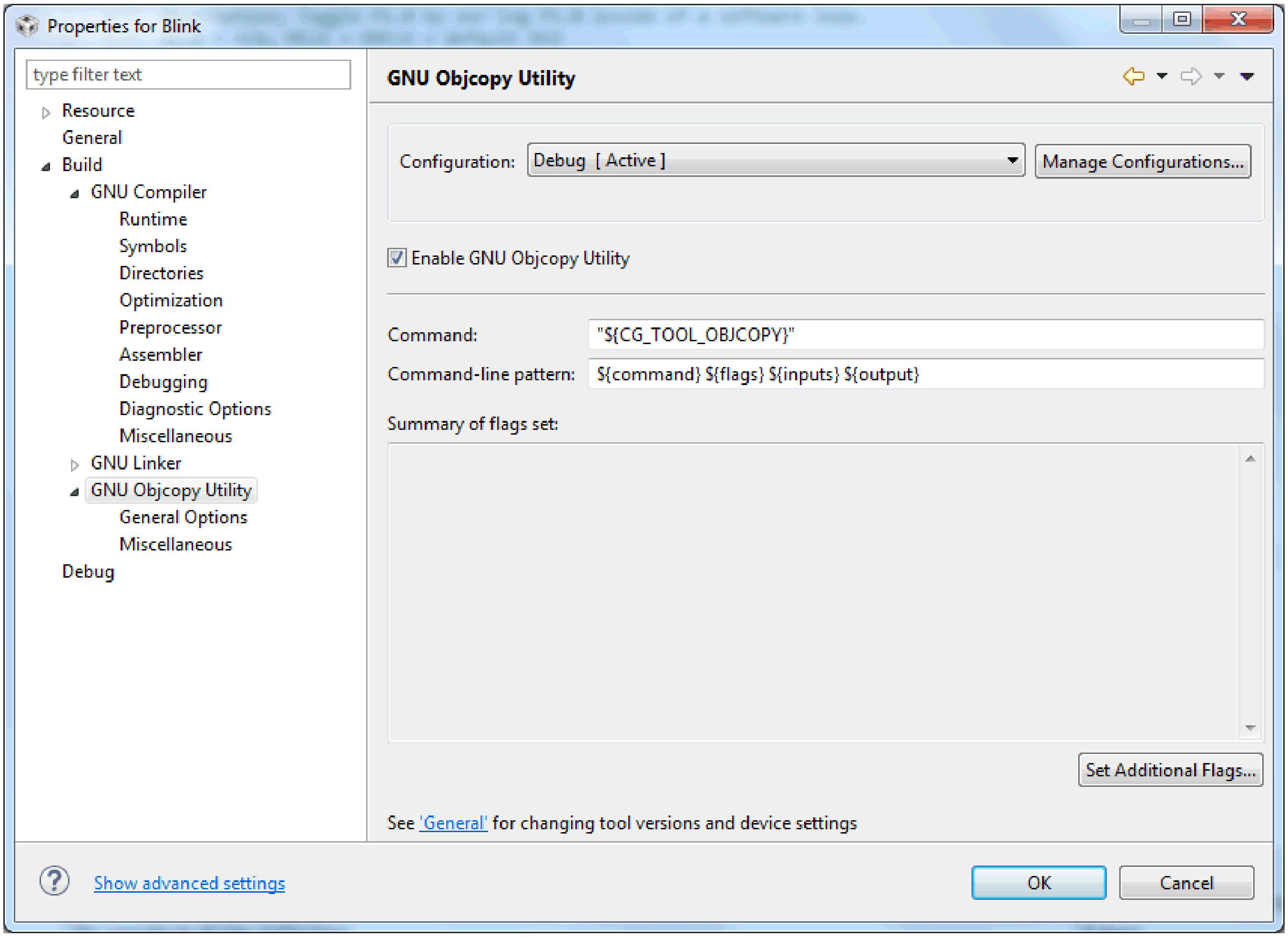
Table 16 describes the options that are available for MSP430 GCC Linker Miscellaneous settings.

Table 16. MSP430 GCC Linker Miscellaneous Settings

Option	Description
Other flags	Specifies individual flags based on the user requirements.

3.3.16 GNU Objcopy Utility

Figure 23 shows the MSP430 GCC GNU Objcopy Utility settings window.



Properties for Blink

GNU Objcopy Utility

Configuration: Debug [Active]

☑ Enable GNU Objcopy Utility

Command: "${CG_TOOL_OBJCOPY}"

Command-line pattern: ${command} ${flags} ${inputs} ${output}

Summary of flags set:

See 'General' for changing tool versions and device settings

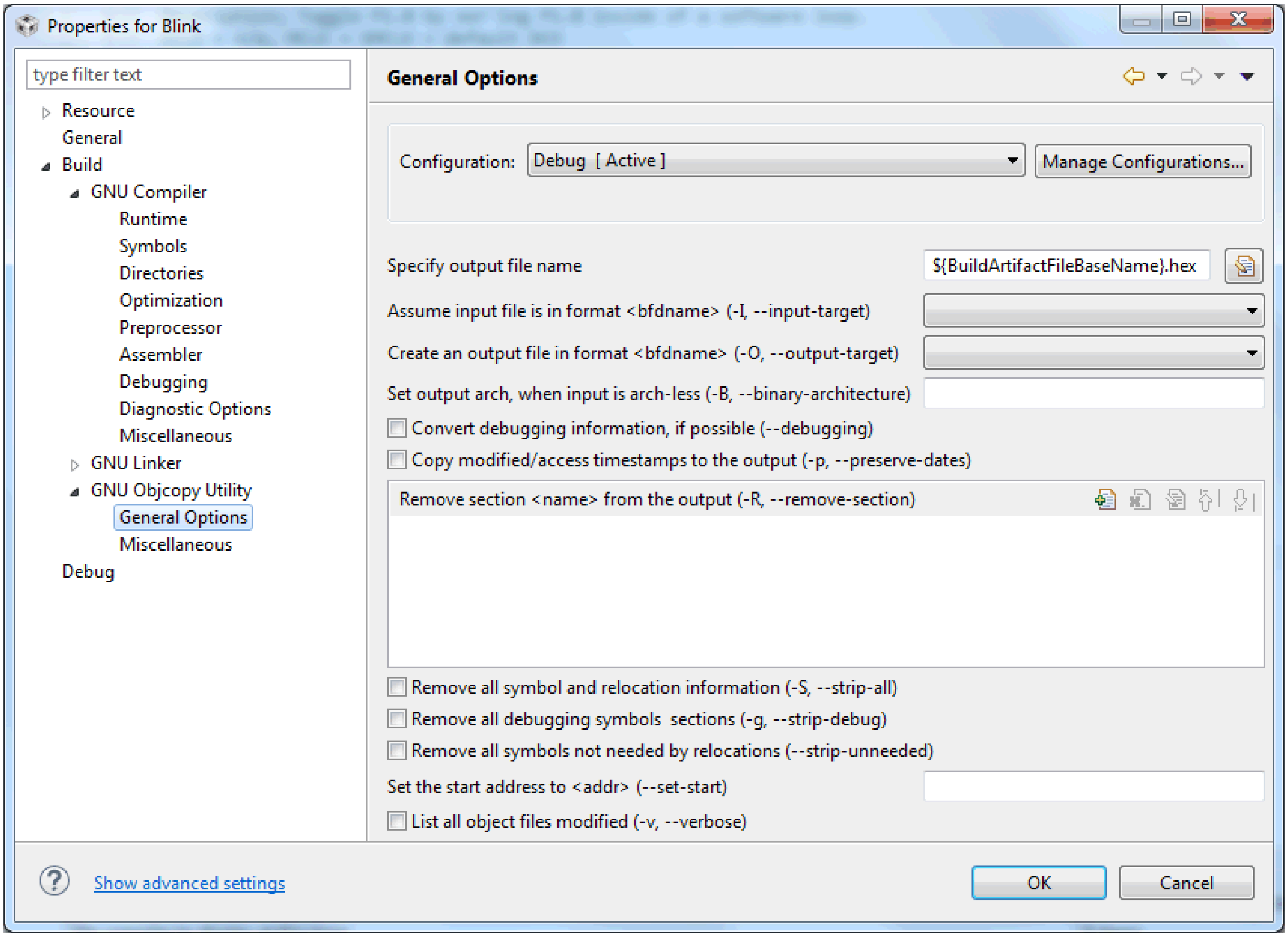
Figure 23. MSP430 GCC GNU Objcopy Utility Settings

Table 17 describes the options that are available for GNU Objcopy Utility.

Table 17. MSP430 GCC GNU Objcopy Utility Settings

Option	Description
Enable GNU Objcopy Utility	Enable this option to enable the GNU Objcopy Utility. It is disabled by default.
Command	GNU Objcopy location
Command-line pattern	Command line parameters
Summary of flags set	Command line with which the GNU Objcopy is called

Figure 24 shows the MSP430 GCC GNU Objcopy Utility General Options settings window.



General Options

Configuration: Debug [Active]

Specify output file name: ${BuildArtifactFileBaseName}.hex

Assume input file is in format <bfdname> (-I, --input-target)

Create an output file in format <bfdname> (-O, --output-target)

Set output arch, when input is arch-less (-B, --binary-architecture)

☐ Convert debugging information, if possible (--debugging)

☐ Copy modified/access timestamps to the output (-p, --preserve-dates)

Remove section <name> from the output (-R, --remove-section)

☐ Remove all symbol and relocation information (-S, --strip-all)

☐ Remove all debugging symbols sections (-g, --strip-debug)

☐ Remove all symbols not needed by relocations (--strip-unneeded)

Set the start address to <addr> (--set-start)

☐ List all object files modified (-v, --verbose)

Figure 24. MSP430 GCC GNU Objcopy Utility General Options Settings

Table 18 describes the options that are available for GNU Objcopy Utility General Options.

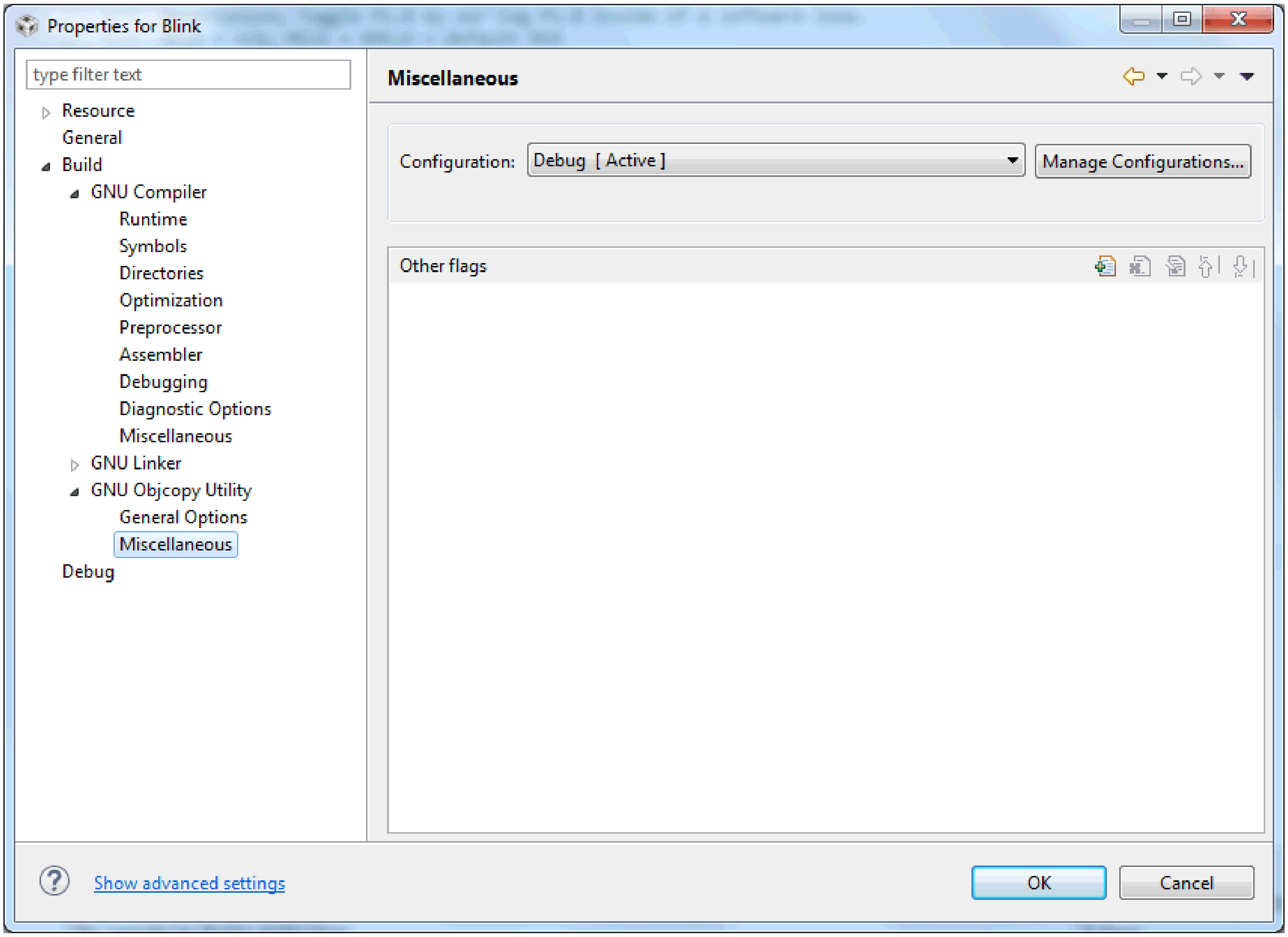
Table 18. MSP430 GCC GNU Objcopy Utility General Options Settings

Option	Description
Specify output file name	Specifies the output file name
Assume input file is in format <bfdname> (-I, --input-target)	-I bfdname --input-target=bfdname Consider the source file's object format to be bfdname, rather than attempting to deduce it.
Create an output file in format <bfdname> (-O, --output-target)	-O bfdname --output-target=bfdname Write the output file using the object format bfdname.
Set output arch, when input is arch-less (-B, --binary-architecture)	-B bfdarch --binary-architecture=bfdarch Useful when transforming a architecture-less input file into an object file. In this case the output architecture can be set to bfdarch.
Convert debugging information, if possible (--debugging)	Convert debugging information, if possible. This is not the default because only certain debugging formats are supported, and the conversion process can be time consuming.
Copy modified/access timestamps to the output (-p, --preserve-dates)	Set the access and modification dates of the output file to be the same as those of the input file.

Table 18. MSP430 GCC GNU Objcopy Utility General Options Settings (continued)

Option	Description
Remove section <name> from the output (-R, --remove-section)	-R sectionpattern --remove-section=sectionpattern Remove any section matching sectionpattern from the output file. This option may be given more than once. Note that using this option inappropriately may make the output file unusable. Wildcard characters are accepted in sectionpattern. Using the -j and -R options together results in undefined behavior.
Remove all symbol and relocation information (-S, --strip-all)	Do not copy relocation and symbol information from the source file.
Remove all debugging symbols sections (-g, --strip-debug)	Do not copy debugging symbols or sections from the source file.
Remove all symbols not needed by relocations (--strip-unneeded)	Strip all symbols that are not needed for relocation processing.
Set the start address to <addr> (--set-start)	Set the start address of the new file to the specified value. Not all object file formats support setting the start address.
List all object files modified (-v, --verbose)	Verbose output: list all object files modified. In the case of archives, 'objcopy -V' lists all members of the archive.

Figure 25 shows the MSP430 GCC GNU Objcopy Utility Miscellaneous settings window.



Configuration: Debug [Active]

Figure 25. MSP430 GCC GNU Objcopy Utility Miscellaneous Settings

Table 19 describes the options that are available for GNU Objcopy Utility Miscellaneous.

Table 19. MSP430 GCC GNU Objcopy Utility Miscellaneous Settings

Option	Description
Other flags	Specifies individual flags based on the user requirements.

3.4 Change an Existing CCS project That Uses TI Compiler to MSP430 GCC

An existing CCS project that uses the TI compiler can be changed to use MSP430 GCC. Refer to **Using MSP430-GCC with CCSv6** (http://processors.wiki.ti.com/index.php/Using_MSP430-GCC_with_CCSv6) for more details.

3.5 Create a New CDT Project Using MSP430 GCC

A standard Eclipse C/C++ project (rather than a CCS project) can use the MSP430 GCC compiler. This would be necessary to debug using GDB instead of the CCS debugger.

To create a new Eclipse C/C++ project that uses MSP430 GCC tools to build the executable, refer to **Using MSP430-GCC with CCSv6** (http://processors.wiki.ti.com/index.php/Using_MSP430-GCC_with_CCSv6).

3.6 GDB With MSP430 and CCSv6

CCSv6 and later can use GDB to debug MSP430 devices. To use the CCS GUI for things like setting and removing breakpoints, the project should be an Eclipse C/C++ project rather than a CCS project.

Refer to **GDB with MSP430 and CCSv6** (http://processors.wiki.ti.com/index.php/GDB_with_MSP430_and_CCSv6) for more details.

3.7 CCS Compared to MSP430 GCC

Some CCS features are not supported in MSP430 GCC. These features are:

- Optimizer Assistant
- ULP Advisor
- Memory Protection Unit and Intellectual Property Encapsulation GUI configuration
- Memory allocation

To gain access to these features, the TI Compiler must be used.

4 MSP430 GCC Stand-Alone Package

4.1 MSP430 GCC Stand-Alone Packages

MSP430 GCC stand-alone package is provided for users who prefer to use MSP430 GCC compiler with other IDE or console-based solutions for compiling and debugging. This stand-alone package supports different operating systems and is provided in different formats:

- GCC and GDB binaries for Windows, Linux, and Mac
- MSP430 header and linker files
- MSP430 GCC source code
- GBD agent configuration

Table 20 lists all the available MSP430 GCC stand-alone packages.

Table 20. MSP430 GCC Stand-Alone Package

Software	Description
msp430-gcc-full-linux-installer-x.x.x.x.run	MSP430 GCC Linux installer incl. support files and debug stack and USB drivers - apply chmod x before executing the package
msp430-gcc-full-windows-installer- x.x.x.x.exe	MSP430 GCC Windows installer incl. support files and debug stack and USB drivers

Table 20. MSP430 GCC Stand-Alone Package (continued)

Software	Description
msp430-gcc-full-mac-installer- x.x.x.x.app	MSP430 GCC Mac installer incl. support files and debug stack and USB drivers
msp430-gcc-linux-installer- x.x.x.x.run	MSP430 GCC Linux installer - compiler only - apply chmod x before executing the package
msp430-gcc-windows-installer- x.x.x.x.exe	MSP430 GCC Windows installer - compiler only
msp430-gcc-mac-installer- x.x.x.x.app	MSP430 GCC Mac installer - compiler only
msp430-gcc-support-files.zip	Header Files
msp430-gcc-source.tar.bz2	MSP430 GCC source files
md5sum.txt	MD5 checksums

4.1.1 MSP430 GCC Stand-Alone Package Folder Structure

The placeholder INSTALL_DIR refers to the directory where you installed the GCC MSP430 package.

- INSTALL_DIR
  - bin
    - MSP430 GCC Compiler binary
    - GDB binary
    - binutils
    - Tcl
    - GDB Agent (http://processors.wiki.ti.com/index.php/XDS_Emulation_Software_Package)
    - MSP430 Debug Stack (http://www.ti.com/tool/mspds)
  - common
  - docs
  - emulation
    - Windows USB-FET Drivers
  - examples
  - include
    - MSP430 Support Files
  - lib
  - libexec
  - msp430-elf
    - lib
      - libatomic
      - libgcc
      - libssp
      - libstdC++-v3
      - libbacktrace
      - libgcc-math
      - libgloss
      - libiberty
      - libsanitizer
      - newlib
  - msp430.dat
  - GCC_xx_manifest.pdf

4.2 Package Content

MSP430 GCC contains binary and sources software development tools for all TI MSP430 devices. The toolchain contains: compiler, assembler, linker, debugger, libraries, and other utilities.

These software development tools include:

- Compiler: MSP430 GCC (**msp430-elf-gcc**) is configured to compile C or C++.
- binutils: assembler, archiver, linker, librarian, and other programs.
- Newlib is the standard C library
- Debugging: gdb_agent_console.exe and gdb_agent_gui.exe
- Source code: Compiler source code is available at http://www.ti.com/tool/msp430-gcc-opensource

4.3 MSP430 GCC Options

The MSP430 GCC toolchain supports the following options:
-msim -masm-hex -mmcu= -mcpu= -mlarge -msmall -mrelax -mhwmult=

For the most up-to-date information, refer to the GCC online documentation at https://gcc.gnu.org/onlinedocs/gcc/MSP430-Options.html.

Table 21. MSP430 GCC Command Options

Option	Description
-masm-hex	Force assembly output to always use hex constants. Normally such constants are signed decimals, but this option is available for test suite or aesthetic purposes.
-mmcu=	Select the MCU to target. This is used to create a C preprocessor symbol based upon the MCU name, converted to upper case and prefixed and postfixed with '__'. This in turn is used by the 'msp430.h' header file to select an MCU-specific supplementary header file. The option also sets the ISA to use. If the MCU name is one that is known to only support the 430 ISA then that is selected, otherwise the 430X ISA is selected. A generic MCU name of 'msp430' can also be used to select the 430 ISA. Similarly, the generic 'msp430x' MCU name selects the 430X ISA. In addition an MCU-specific linker script is added to the linker command line. The script's name is the name of the MCU with '.ld' appended. Thus specifying '-mmcu=xxx' on the gcc command line defines the C preprocessor symbol __XXX__ and cause the linker to search for a script called 'xxx.ld'. This option is also passed on to the assembler.
-mwarn-mcu -mno-warn-mcu	This option enables or disables warnings about conflicts between the MCU name specified by the -mmcu option and the ISA set by the -mcpu option or the hardware multiply support set by the -mhwmult option. It also toggles warnings about unrecognized MCU names. This option is on by default.
-mcpu=	Specifies the ISA to use. Accepted values are 'msp430', 'msp430x' and 'msp430xv2'. This option is deprecated. The '-mmcu=' option should be used to select the ISA.
-msim	Link to the simulator runtime libraries and linker script. Overrides any scripts that would be selected by the '-mmcu=' option.
-mlarge	Use large-model addressing (20-bit pointers, 32-bit size_t).
-msmall	Use small-model addressing (16-bit pointers, 16-bit size_t).
-mrelax	This option is passed to the assembler and linker, and allows the linker to perform certain optimizations that cannot be done until the final link.

Table 21. MSP430 GCC Command Options (continued)

Option	Description
-mhwmult=	Describes the type of hardware multiply supported by the target. Accepted values: • 'none' for no hardware multiply • '16bit' for the original 16-bit-only multiply supported by early MCUs • '32bit' for the 16/32-bit multiply supported by later MCUs • 'f5series' for the 16/32-bit multiply supported by F5-series MCUs. • 'auto' can also be given. This tells GCC to deduce the hardware multiply support based upon the MCU name provided by the '-mmcu' option. If no '-mmcu' option is specified, then '32bit' hardware multiply support is assumed. 'auto' is the default setting. Hardware multiplies are normally performed by calling a library routine. This saves space in the generated code. When compiling at '-O3' or higher, however, the hardware multiplier is invoked inline. This makes for larger but faster code. The hardware multiply routines disable interrupts while running and restore the previous interrupt state when they finish. This makes them safe to use inside interrupt handlers as well as in normal code.
-minrt	Enable the use of a minimum runtime environment (no static initializers or constructors). This is intended for memory-constrained devices. The compiler includes special symbols in some objects that tell the linker and runtime which code fragments are required.
-mcode-region= -mdata-region=	These options tell the compiler where to place functions and data that do not have one of the lower, upper, either or section attributes. Possible values are: • lower • upper • either • any The first three behave like the corresponding attribute. The fourth possible value ('any') is the default. It leaves placement entirely up to the linker script and how it assigns the standard sections (.text, .data, and so on) to the memory regions.
-msilicon-errata=NAME[,NAME...]	Implements fixes for named silicon errata. Multiple silicon errata can be specified by multiple uses of the -msilicon-errata option or by including the errata names, separated by commas, on an individual -msilicon-errata option. Errata names currently recognized by the assembler are: cpu4 = PUSH #4 and PUSH #8 need longer encodings on the MSP430. This option is enabled by default, and cannot be disabled. cpu8 = Do not set the SP to an odd value. cpu11 = Do not update the SR and the PC in the same instruction. cpu12 = Do not use the PC in a CMP or BIT instruction. cpu13 = Do not use an arithmetic instruction to modify the SR. cpu19 = Insert NOP after CPUOFF.
-msilicon-errata-warn=NAME[,NAME...]	Like the -msilicon-errata option, except that instead of fixing the specified errata, a warning message is issued instead. This option can be used with -msilicon-errata to generate messages whenever a problem is fixed, or on its own to inspect code for potential problems.

4.4 MSP430 Built-in Functions

GCC provides special built-in functions to aid in the writing of interrupt handlers in C.

__bic_SR_register_on_exit (int mask)

This clears the indicated bits in the saved copy of the status register that currently resides on the stack. This only works inside interrupt handlers and the changes to the status register only take affect after the handler returns.

__bis_SR_register_on_exit (int mask)

This sets the indicated bits in the saved copy of the status register that currently resides on the stack. This only works inside interrupt handlers and the changes to the status register only take affect after the handler returns.

4.5 MSP430 GCC Interrupts Definition

To define an interrupt using MSP430 GCC, use the following syntax:

```
void __attribute__ ((interrupt(INTERRUPT_VECTOR))) INTERRUPT_ISR (void)
```

The "static" keyword should not be used on ISR definition.

Code Example:

```
void __attribute__ ((interrupt(UNMI_VECTOR))) UNMI_ISR (void)
{ // isr }
```

You could also use the macro defined in the iomacros.h file:

```
#define __interrupt_vec(vec)__attribute__((interrupt(vec)))
```

Example:

```
void __interrupt_vec(UNMI_VECTOR) UNMI_ISR (void)
{}
```

4.6 Quick Start: Blink the LED

This document assumes that a version of the GNU Make utility is installed on the system and that it is available on the system path. The placeholder **INSTALL_DIR** refers to the directory where the GCC MSP430 package is installed. The directory **INSTALL_DIR/bin** should be on the system path.

4.6.1 Building With a Makefile

1. In the command terminal, go to the **INSTALL_DIR\examples** directory.
2. There are examples for Windows and Linux. They are located in the corresponding subdirectories. Choose one of the examples suitable for the operating system and MSP430 target device.
3. Change to the directory and type **make**.
4. The binary can now be downloaded and debugged on the target hardware.

4.6.2 Building Manually With gcc

To build one of the examples manually, open a terminal and change to the example suitable for the target device and operating system. The compiler executable **msp430-elf-gcc** must be available on your system path.

```
msp430-elf-gcc -I <Path to MSP430 Support Files> -L <Path to MSP430 Support Files>
-T DEVICE.ld -mmcu=DEVICE -O2 -g blink.c -o blink.o
```

The placeholder **<Path to MSP430 Support Files>** is the directory that contains the MSP430 support files (header files and linker scripts to support the different MSP430 devices).

The placeholder **DEVICE** tells the compiler and linker to create code for the target device. The command line argument -T DEVICE.ld is optional, as the compiler automatically selects the correct linker script from the -mmcu=DEVICE argument.

Example

```
msp430-elf-gcc -I=../../../include -L=../../../include -T msp430fr5969.ld -mmcu=msp430fr5969 -O2 -
g blink.c -o blink.o
```

4.6.3 Debugging

4.6.3.1 Starting GDB Agent

On Microsoft Windows, the GDB Agent is available as either as a small GUI application or on the command line. On GNU Linux, only the command line version is available.

4.6.3.1.1 Using the GUI

Open the **INSTALL_DIR/bin** directory and double-click gdb_agent_gui.

1. After the program starts, click the button **Configure**, select **msp430.dat**, and click **Open**.
2. Click on the button **Start** under the Panel Controls.
3. The "Log" window now contains the status message "Waiting for client".
4. Leave the window open until the end of the debugging process.

4.6.3.1.2 Using the Command Line

Open a command terminal, change to INSTALL_DIR and type:

Linux

```
./bin/gdb_agent_console msp430.dat
```

Windows

```
.\bin\gdb_agent_console msp430.dat
```

4.6.3.2 Debugging With GDB

4.6.3.2.1 Running a Program in the Debugger

1. In the command terminal, go to the INSTALL_DIR\examples\[Selected example], and type the command **make debug**.
2. This command starts the GDB and waits for commands. This is indicated by the prompt <gdb>.
3. To connect GDB to the GDB Agent, type the command **target remote :55000** and press enter.
4. To load a program binary to the MSP430 target device, type **load**.
5. Type the command **continue** (short version: c) to tell GDB to run the loaded program.
6. The LED on the target board blinks.

4.6.3.2.2 Setting a Breakpoint

1. Connect the GDB to the GDB Agent as previously described and load a program to the device.
2. To set a breakpoint on a function, type **break function name**.
3. To set a breakpoint on a source line, type **break filename:line**.
4. When you run the program, the program execution stops at the entry to the specified function or stops at the specified line.

4.6.3.2.3 Single Stepping

1. Connect the GDB to the GDB Agent as previously described and load a program to the device.
2. After the debugger has stopped the program at a breakpoint, you can step through the code:
   (a) To execute the source line, type **next**. **next** does not step into functions, it executes the complete function and stops on the line following the function call.
   (b) To execute the next source line and step into functions, type **step**.
   (c) To execute the next instruction, type **nexti**.
   (d) To execute the next instruction and step into functions, type **stepi**.

4.6.3.2.4 Stopping or Interrupting a Running Program

1. Connect the GDB to the GDB Agent as previously described and load a program to the device.
2. To stop a running program and get back to the GDB command prompt, type **Ctrl+C**. This currently applies only on Linux.

4.6.4 Creating a New Project

1. Create a directory for your project.
2. Copy one of the example project makefiles into the project directory.
3. Open the copied makefile and set the variable DEVICE to the target device.
4. Set the variable GCC_DIR to point to the directory where the GCC MSP430 package is installed.
5. Include all of the project source files (that is, the *.c files) as a dependency for the first target of the makefile.
6. Go to the project directory in a terminal and type **make** to build the project or **make debug** to start debugging the project.

```
OBJECTS=blink.o

GCC_DIR =  ../../../bin
SUPPORT_FILE_DIRECTORY = ../../../include

# Please set your device here
DEVICE  = msp430X
CC      = $(GCC_DIR)/msp430-elf-gcc
GDB     = $(GCC_DIR)/msp430-elf-gdb

CFLAGS = -I $(SUPPORT_FILE_DIRECTORY) -mmcu=$(DEVICE) -O2 -g
LFLAGS = -L $(SUPPORT_FILE_DIRECTORY) -T $(DEVICE).ld

all: ${OBJECTS}
    $(CC) $(CFLAGS) $(LFLAGS) $? -o $(DEVICE).out

debug: all
    $(GDB) $(DEVICE).out
```

4.7 GDB Settings

The GDB Agent is a tool to connect the GDB with the target hardware to debug software. The GDB Agent uses the MSP430 debug stack to connect to the hardware and provides an interface to the GDB. On Windows, both a console and a GUI application version of the GDB agent are provided. Only the console application is supported on Linux.

4.7.1 Console Application

If you use the console application, invoke it from a command terminal using following syntax:

Linux

INSTALL_DIR/bin/gdb_agent_console INSTALL_DIR/msp430.dat

Windows

INSTALL_DIR\bin\gdb_agent_console INSTALL_DIR\msp430.dat

The console application opens a TCP/IP port on the local machine. It displays the port number in the console. By default, this port number is 55000.

4.7.2 GUI Application

After you start the GUI application, configure the GUI and then start the GDB server. For more information, refer to the XDS GDB Agent online documentation at http://processors.wiki.ti.com/index.php/XDS_GDB_Agent.

1. Click the **Configure** button and, in the Select board configuration file window, select the msp430.dat file. If successfully configured, an MSP430 device is displayed in the **<Targets>** list. The TCP/IP port for the GDB server is displayed when the MSP430 device is selected from the list.
2. To start the GDB Agent, click the **Start** button when the MSP430 device is selected.

4.7.3 Attaching the Debugger

After starting the debugger and to attach to the GDB server, use the target remote **[<host ip address>]:<port>** command, where **<port>** is the TCP/IP port from above. If the GDB Agent runs locally, omit the host IP address.

4.7.4 Configuring the Target Voltage

To configure the target voltage for the device, open the file msp430.dat in a text editor. To change the voltage, modify the key msp430_vcc. By default, this value is set to 3.3 volts.

4.7.5 Resetting the Target

To reset the target, use the **monitor reset** command.

4.7.6 Halting the Target

To halt the target, use the **monitor halt** command.

5 Building MSP430 GCC From Sources

5.1 Required Tools

This document assumes that the required tools are installed on the system and that they are available on the system path.

- GNU make
- GCC and binutils
- bzip2 and tar

5.2 Building MSP430 GCC

NOTE: The following Linux and Windows builds are from Linux (cross compile Windows). The OS X builds are native.

Make sure that LIBRARY_PATH is set correctly.

The building steps for Linux, Windows, and Mac (see Section 5.2.1, Section 5.2.2, and Section 5.2.3, respectively) are similar to each other.

1. Download and extract the MSP430 GCC sources from TI website.
2. **export**: Export all needed variables
3. **mkdir build**: Create a build directory
4. **cd build**: Change to build directory
5. **../sources/tools/configure**: Configure the entire GCC sources directory before the build. See the corresponding options for every build below.
6. **make all**: Compile the entire program.
7. **make install**: Copy the executables, libraries, and so on to the file names where they should reside for

actual use.

8. **make info**: Generate Compile Info files.
9. **make install-info**: Generate install Info files.

NOTE:

1. MSP430 GCC must always build in a separate (empty) directory.
2. Make sure that you have all the required packages in your system (libgmp, libmpfr, zlib, tcl, tk, itlc).

5.2.1 Building MSP430 GCC on Linux

```
% wget http://software-
dl.ti.com/msp430/msp430_public_sw/mcu/msp430/MSPGCC/latest/exports/msp430-gcc-
source.tar.bz2
% tar xvjf msp430-gcc-source.tar.bz2?tracked=1
% mkdir build
% cd build
% ../sources/tools/configure --prefix=/path/to/install_location --target=msp430-elf --
enable_languages=c,c++
% make all
% make install
% make info
% make install-info
```

5.2.2 Building MSP430 GCC on Windows (Cross Compiled on Linux)

```
% wget http://software-
dl.ti.com/msp430/msp430_public_sw/mcu/msp430/MSPGCC/latest/exports/msp430-gcc-
source.tar.bz2
% tar xvjf msp430-gcc-source.tar.bz2?tracked=1

% yum install mingw32-gcc mingw32-gcc-g++
  Configm win32 gcc:
  % ls -1 /usr/bin/*gcc
  /usr/bin/gcc
  /usr/bin/i386-redhat-linux-gcc
  /usr/bin/i686-pc-mingw32-gcc

% export PATH=/path/to/install_location/bin:$PATH
% mkdir build
% cd build

From above yum install, the "i686-pc-mingw32" matches --
host below.   You must have a pre-built msp430-elf native toolchain in your $PATH:

% ../sources/tools/configure --prefix=/path/to/install_location --target=msp430-elf --
enable-languages=c,c++ --host=i686-pc-mingw32 --disable-itcl --disable-tk --disable-
tcl --disable-libgui --disable-gdbtk
% make all
% make install
% make info
% make install-info
```

5.2.3 Building MSP430 GCC on OS X

```
%  wget http://software-
dl.ti.com/msp430/msp430_public_sw/mcu/msp430/MSPGCC/latest/exports/msp430-gcc-
source.tar.bz2%  tar xvjf msp430-gcc-source.tar.bz2?tracked=1

% export PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin
% export LIBRARY_PATH=/usr/X11/lib
% export MACOSX_DEPLOYMENT_TARGET=10.5
% mkdir build%  cd build% ../sources/tools/configure --target=msp430-elf --
prefix=/path/to/install_location --enable-languages=c,c++ --disable-werror --disable-
itcl --disable-tk --disable-tcl --disable-libgui --disable-gdbtk --disable-sim
% make all
% make install
% make info
% make install-info
```

5.2.4 Building MSP430 GCC Stand-Alone Full Package

1. MSP430 GCC Compiler
   (a) Download the MSP430 GCC Installer Compiler only from http://www.ti.com/tool/msp430-gcc-opensource
   (b) Use the generated MSP430 GCC version (see Section 5.2)
2. USB driver package (Windows only)
   (a) Download "Stand-alone Driver Installer for USB Low-Level Drivers" from http://www.ti.com/tool/mspds
3. MSPDS OS package
   (a) Download "MSP Debug Stack Open Source Package" from http://www.ti.com/tool/mspds
4. Build MSPDebugStack
   (a) Extract "MSP Debug Stack Open Source Package"
   (b) Follow the instructions in "README-BUILD.txt"
5. GDB agent
   (a) Download the GDB agent from http://processors.wiki.ti.com/index.php/XDS_Emulation_Software_Package
6. MSP430 support files for GCC
   (a) Download "msp430-gcc-support-files.zip" from http://software-dl.ti.com/msp430/msp430_public_sw/mcu/msp430/MSPGCC/latest/index_FDS.html

6 MSP430 GCC and MSPGCC

The new GCC compiler for MSP low-power microcontrollers conforms to the MSP Embedded Application Binary Interface (EABI) (see SLAA534). This allows GCC to interoperate with the proprietary TI compiler.

For example, assembly functions can be written in the same way, and libraries that are built with one compiler can be used as part of executables built with the other compiler. Aligning with the MSP EABI required breaking compatibility with the prior MSPGCC compiler. This document gives a brief overview of the ABI changes that are most likely to be noticed by and to affect a developer who is moving from MSPGCC to the newer GCC compiler for MSP.

6.1 Calling Convention

For developers writing assembly code, the most noticeable part of an ABI is the calling convention. Full specification of the calling convention is very detailed (see *MSP430 Embedded Application Binary Interface* SLAA534), but developers writing assembly do not typically use most of it. There are three basic differences between MSPGCC and the GCC compiler for MSP in the calling convention that are important to be aware of:

- In MSPGCC, registers are passed starting with R15 and descending to R12. For example, if two integers are passed, the first is passed in R15 and the second is passed in R14. In contrast, the MSP430 EABI specifies that arguments are passed beginning with R12 and moving up to R15. So, in the same situation, registers R12 and R13 would hold the two arguments. In both cases, after the registers R12 through R15 are used, continued arguments are passed on the stack. If you are using stack-based arguments, you should consult the EABI specification.

- MSPGCC and the GCC compiler for MSP use different registers for the return value. MSPGCC places the return value in R15 (or R15 and consecutive lower registers if the value is larger than a word), while the EABI specifies that the return value is placed in R12.
- In MSPGCC, register R11 is considered a save on entry register and needs to be saved and restored by the callee if it is used in the called function. Conversely, the MSP EABI specifies that R11 is a save on call register, so it needs to be saved and restored by the calling function if its value will be needed after a function call. For comparison purposes, R4 to R10 are save on entry registers for both compilers, and R12 to R15 are save on call.

These are the key differences to be aware of when moving between the compilers. If you are writing assembly code that passes parameters on the stack or that passes structures by value, you should consult the MSP EABI document for additional information.

6.2 Other Portions of the ABI

Many other pieces make up the EABI, such as the object file format; debug information, and relocation information that is used when linking together files. However, in general, these pieces do not affect migration.

One other area to be aware of is that the details of data layout differ between ABIs. If you are relying on advanced data layout details such as layout of structures and bitfields, consult the MSP EABI document (SLAA534).

7 Appendix

7.1 GCC Intrinsic Support

The GCC Compiler supports the same intrinsics that the TI CGT for MSP430 does. These are:

1. unsigned short __**bcd_add_short**(unsigned short op1, unsigned short op2);
2. unsigned long __**bcd_add_long**(unsigned long op1, unsigned long op2);
3. unsigned short __**bic_SR_register**(unsigned short mask); BIC mask, SR
4. unsigned short __**bic_SR_register_on_exit**(unsigned short mask);
5. unsigned short __**bis_SR_register**(unsigned short mask);
6. unsigned short __**bis_SR_register_on_exit**(unsigned short mask);
7. unsigned long __**data16_read_addr**(unsigned short addr);
8. void __**data16_write_addr** (unsigned short addr, unsigned long src);
9. unsigned char __**data20_read_char**(unsigned long addr);
10. unsigned long __**data20_read_long**(unsigned long addr);
11. unsigned short __**data20_read_short**(unsigned long addr);
12. void __**data20_write_char**(unsigned long addr, unsigned char src);
13. void __**data20_write_long**(unsigned long addr, unsigned long src);
14. void __**data20_write_short**(unsigned long addr, unsigned short src);
15. void __**delay_cycles**(unsigned long);
16. void __**disable_interrupt**(void); AND __**disable_interrupts**(void);
17. void __**enable_interrupt**(void); AND __**enable_interrupts**(void);
18. unsigned short __**get_interrupt_state**(void);
19. unsigned short __**get_SP_register**(void);
20. unsigned short __**get_SR_register**(void);
21. unsigned short __**get_SR_register_on_exit**(void);
22. void __**low_power_mode_0**(void);
23. void __**low_power_mode_1**(void);
24. void __**low_power_mode_2**(void);
25. void __**low_power_mode_3**(void);
26. void __**low_power_mode_4**(void);
27. void __**low_power_mode_off_on_exit**(void);
28. void __**no_operation**(void);
29. void __**set_interrupt_state**(unsigned short src);
30. void __**set_SP_register**(unsigned short src);
31. unsigned short __**swap_bytes**(unsigned short src);

7.2 GCC Function Attribute Support

1. **interrupt(x)**

   Make the function an interrupt service routine for interrupt "x".
2. **signal**

   Make an interrupt service routine allow further nested interrupts.
3. **wakeup**

   When applied to an interrupt service routine, wake the processor from any low-power state as the routine exits. When applied to other routines, this attribute is silently ignored.
4. **naked**

   Do not generate a prologue or epilogue for the function.
5. **critical**

   Disable interrupts on entry, and restore the previous interrupt state on exit.
6. **reentrant**

   Disable interrupts on entry, and always enable them on exit.
7. **noint_hwmul**

   Suppress the generation of disable and enable interrupt instructions around hardware multiplier code.

8 References

1. Using the GNU Compiler Collection, Richard M. Stallman (http://gcc.gnu.org/onlinedocs/gcc.pdf): refer to "MSP430 Options" section.
2. GDB: The GNU Project Debugger, Free Software Foundation, Inc. (https://sourceware.org/gdb/current/onlinedocs/)

Revision History

Changes from September 15, 2015 to March 29, 2016 **Page**

- Added -mwarn-mcu row in Table 21, *MSP430 GCC Command Options* 35
- Added -msilicon-errata row in Table 21, *MSP430 GCC Command Options* 36
- Added -msilicon-errata-warn row in Table 21, *MSP430 GCC Command Options* 36
- Added Section 4.5, *MSP430 GCC Interrupts Definition* 37
- Added example in Section 4.6.2, *Building Manually With gcc* 37
- Changed step (1) in Section 4.6.3.2.1, *Running a Program in the Debugger* 38
- Added "at the entry to the specified function or stops at the specified line" to step (4) of Section 4.6.3.2.2, *Setting a Breakpoint* 38
- Added "Open the copied makefile" to step (3) in Section 4.6.4, *Creating a New Project* 39
- Added code example in Section 4.6.4, *Creating a New Project* 39
- Added note at end of Section 5.2, *Building MSP430 GCC* 41

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

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 relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license 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 significant portions 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. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265