Building and Editing with MULTI® 2000

UNIX®

DISCLAIMER

GREEN HILLS SOFTWARE, INC. MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. Further, Green Hills Software, Inc. reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Green Hills Software, Inc. to notify any person of such revision or changes.

PubID: M41W-A1299-2BNG

Time Stamp: December 20, 1999 3:41 pm

CONTENTS

CONTENTS

CONTENTS

CONTENTS

Preface

***This chapter contains*:**

- About the MULTI manuals
- Conventions

About the MULTI manuals

This manual systematically documents all the features and commands of the MULTI Builder and Editor. The comprehensive index will help you locate the information you need.

For other components of MULTI, such as the Debugger, refer to the *Debugging with AdaMULTI 2000* manual.

For specific target systems, refer to the *Development Guide* for your target.

Conventions

Typographical conventions

Convention	Example	Description
italic text in a command line	**-o** *filename*	place-holder for mandatory user-supplied arguments
square brackets, []	**.macro** *name* [*list*]	encloses optional commands, terms, or arguments
square brackets [] around boldface word "**default"**	Specifies **char** as signed. **[default]**	command or option is the default
menu > item > sub-item...	File > Open...	menu bar, menu items, sub-menu items...
Enter *something*	Enter **adamulti a.out**	Type *something* AND press the Enter key. Compare with "Type *something*" below.
Type *something*	Type foo.s and press Edit	Type *something* WITHOUT pressing the Enter key. Compare with "Enter *something*" above.

For example, in the command description:

gxyz [-*processor*] *filename*

the command **gxyz** should be entered as given, the word *processor* may optionally be substituted with an appropriate option, and the word *filename* must be replaced with an appropriate file name.

GUI mode conventions

The main MULTI windows in the Builder, Editor, and Debugger contain some or all of the following regions:

Convention	Description
source pane	The portion of the window in which the source code is displayed.
status bar	Displays information, such as the process state and the name of the file being debugged.
command pane	Area to enter commands and display results.
toolbar	Contains buttons for commonly used commands.

GUI conventions

MULTI documentation assumes you have a working knowledge of your operating system and its conventions, including its command-line and GUI interfaces—for example, how to use a mouse and standard menus and commands, and how to open, save, and close files, etc.

Convention	Meaning
First mouse button	Mouse buttons are numbered from the left. The first mouse button is the left-most mouse button.
Shift+Click	Hold down the Shift key while clicking a mouse button.
Ctrl+Click	Hold down the Ctrl key while clicking a mouse button.

Check box conventions

There are two types of check boxes: two-way and three-way.

A two-way check box has two states: either enabled (with a check mark in it) or disabled (when it's empty). For example, Config > Options... > Colors tab > Build file coloring.

A three-way check box has three states (for example, Builder > Project > Options > General tab > Automatically use MVC):

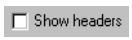
- The first state is On. The box has a plus sign (+), indicating that the option is turned on, overriding any previous or inherited settings.

- The second state is Off. The box contains a minus sign (-), indicating that the option is turned off, overriding any previous or inherited settings.

- The third state is Default. The box is empty, indicating that the inherited state, if any, is used.

Chapter

1

Introduction to MULTI

This chapter contains:

- Features
- Embedded programming in MULTI
- Running MULTI from the command line
- Resources

MULTI is a complete interactive software development environment for programs written in Ada, C, C++, Pascal, and FORTRAN, as well as in assembly language for each supported target. Source code from these languages can be compiled and linked into a single executable in virtually any combination.

> **NOTE**: If you are upgrading from version 1.8.9 to MULTI 2000, **DO NOT INSTALL YOUR MULTI 2000 IN THE SAME LOCATION AS YOUR 1.8.9 RELEASE**.

Features

Some of MULTI's powerful features include:

Project Management

- A **Program Builder** for creating, assembling, and controlling your programming projects. See Chapter 2, "Using the Builder".
- A **Progress Window** to keep you informed at all times as you build your project. See "The Progress window" on page 126.

Version Control

- An automatic **Version Control System** with features for managing revision levels and program branches, and for tracking the origins of suspicious code. See Chapter 4, "Version control".
- The capability to **Merge two or three versions** of a file. See "Viewing inherited options" on page 21.
- **Highlighted Diff Windows** to see the difference between two files. See "Comparing files" on page 160.

Editing

- A built-in **Editor** that is fully configurable, enhanced with special features to support some of the advanced capabilities of MULTI. See Chapter 5, "Using the Editor".

Debugging

For information about the following features, see the *Debugging with AdaMULTI 2000* manual.

- A **Source Level Debugger** that supports mixed language debugging and all C++ and Ada language constructs.
- A **Profiler** that collects data, provides reports, annotates the source code to find hot spots in your program, and provides mechanisms to feed information back into the development process.
- **Run-Time Error Checking** for different classes of errors, implemented with a combination of compiler checks, libraries, and debugger commands.
- **Expression Evaluation** to determine whether your expressions are correct.
- A **Data Explorer** to monitor variables and evaluate expressions during debugging.
- **Memory Leak Detection** to find chunks of memory that have been allocated but are no longer used.
- **Conditional Breakpoints** that cause a breakpoint to be active under conditions you specify.
- A graphical **Ada 95 Type Inheritance** and **C++ Class Browser** to delineate the structure of your classes and of classes you inherit.

Embedded programming in MULTI

MULTI supports embedded development for the following 32- and 64-bit microprocessor families:

Processor families supported by MULTI
680x0/683xx
ARM / Thumb
ColdFire
i960
MCore
MIPS
PowerPC
RH32
SPARC
SH
TriCore
V800
x86 / Pentium

Embedded programming is the programming of microprocessors which are incorporated into an embedded product. Workstations and PCs are used as host computers on which programs are edited and compiled. The programs are then downloaded into a target system to be debugged and executed.

MULTI interfaces to embedded targets by connecting to a debug server. The debug server may reside on the same host as MULTI, or on any other host on your network. The debug server communicates with the target under

development. Green Hills supplies servers for many common target systems and real time operating systems:

- **Instruction set simulators**: Simulators can test programs before target hardware is ready and are available for most processor models. Instruction set simulators incorporate an integrated debug server as a front end.
- **ROM Monitors**: Monserv and the ROM monitor specific to your target support basic debug features, host I/O, a command window, and profiling.
- **In-Circuit Emulators**: Available for several popular emulator families. Emulator servers use your network to communicate with the command interface of the emulator.
- **ROM Emulator**: NetROM provides debugging capabilities with only a single connection to the ROM socket.
- **RTOS (real-time operating system) servers**: Available for several real time operating systems including INTEGRITY from Green Hills Software, ThreadX from Express Logic, OSE from Enea Systems, and VxWorks from Wind River. RTOS servers use ethernet and serial communication to communicate with a debug process running under the RTOS. Commands from MULTI's various debug windows are combined into a single command stream by the RTOS server; the debug process interprets these messages and performs the proper action on the appropriate task.

MULTI allows you to use the same tools for both embedded and native development. The same MULTI program can debug both native and embedded code; the only difference is that MULTI uses a different host processor when communicating with an embedded target.

Running MULTI from the command line

When you start MULTI, it attempts to use the host system windowing package by default. If you start MULTI on a color monitor, it defaults to color. If you start MULTI from a non-windowing monitor or if MULTI encounters problems with the window interface, it comes up in non-GUI mode. If MULTI is incorrectly coming up in non-GUI mode, check that the **DISPLAY** environment variable is set, or set it from the command line with the **-display** option.

If MULTI is in your path, then the command line syntax is:

adamulti [*options*] [*filename*]

If *filename* is a build file, then the Builder starts up with the build file loaded. Note that some options are specific to the Debugger and are not applicable to the Builder.

If *filename* is an executable program file that has had some (or all) of its component modules compiled for debugging (with Green Hills compiler's **-G** or **-g** options), then the Debugger starts up. For a list of command line options to use when opening the Debugger, see "Command line options" on page 7.

If you specify a build file, it can either be a main project or a subproject. To open a subproject directly with the inherited options from a particular main project, specify the main project name followed by the subproject's name in quotes. For example:

```
adamulti "main.bld sub.bld"
```

This opens the subproject, **sub.bld**, with the options inherited from **main.bld**.

See the following table of examples.

How to open MULTI	
Example	**Description**
adamulti	Opens the Builder on the last build file that was open.
adamulti default.bld	Opens the Builder on **default.bld**. If **default.bld** is not found, MULTI will create it.
adamulti foo.bld	Opens the Builder on the file **foo.bld**. The build file may be a main project or a subproject.
adamulti "parent.bld child.bld"	Opens the Builder on the subproject **child.bld** directly with the inherited options from the main project **parent.bld**.
adamulti a.out	Opens the Debugger on the executable **a.out**
adamulti -remote simppc	Opens the Builder and connects it to the simulator **simppc**. In this syntax, it is a function of the debug server whether the Builder window or the Debugger window is opened. See the example below with **adamulti -remote 5emon**.
adamulti -remote simppc a.out	Opens the Debugger on the executable **a.out** and the Debugger is connected to the simulator **simppc**.
adamulti -remote 5emon	Opens the Debugger and connects it to the debug server **5emon**. In this syntax, it is a function of the debug server whether the Builder window or the Debugger window is opened. See the example above with **adamulti -remote simppc**.
adamulti foo.c	Opens the Editor on the file **foo.c**.

Command line options

When you start MULTI from the command line on an executable program file (i.e. when you want to use the Debugger directly), the following options may be used. Some of these options should not be used when starting MULTI on a build file, or when starting MULTI without a file.

-c *file*
: Reads configuration information from file.

-C ***corefile***
: Sets core file. *corefile* is assumed to be a core image of *objectfile*.

-D
: Ignores all currently specified alternate directories.

-data *offset*
: Offsets for all data addresses. This is for position independent data. The offset is entered in decimal by default. A hexadecimal number may be specified by preceding the number with **0x**. For example, **0x10000**.

-dotciscxx
: Treats files ending in **.c** as C++ files instead of C files.

-e *entry*
: Specifies entry label. The default is **main**. In C++ mode, the entry must be specified in such a way that it may be demangled.

-E *file*
: Tells MULTI to debug more than one file. Use this option for each file you wish to debug at the same time. For example, if you want to debug **foo**, **bar**, and **rin**, then type:

```
adamulti foo -E bar -E rin
```

-help
: Runs MULTI and opens the on-line help system with the MULTI manual.

-I *directory*
: Names an alternate directory where files are searched for. Alternate directories are searched in the order given. If a file is not found in an alternate, the current directory is searched.

-L[cpfC]
: Sets language type (C, Pascal, FORTRAN, or C++ respectively). By default, MULTI uses the file name extension to determine the language.

-m *file*
: Uses *file* as default specification file. See “Specification file” on page 8 for more information.

-nocfg

Does not read any of the **.cfg** files of MULTI on startup.

-norc

Does not run any .rc files on startup.

-noshared

Does not debug shared libraries.

-nosplash

Does not open the About banner. See “About MULTI...” on page 41 for more information.

-p *file*

Startup with command playback from *file*.

-P *pid*

Attaches to process with process id *pid*. This option is currently for Solaris only.

-r *file*

Startup with commands recording to *file*.

-R *file*

Startup with commands and output recording to *file*.

-rc *file*

Reads *file* as a command script when the first debugger window appears. The file is read after the global and user script files.

-remote *target*

Attaches to remote debug server with name *target*.

-text *offset*

Offsets for all text addresses. This is for position independent code. The offset is entered in decimal by default. A hexadecimal number may be specified by preceding the number with **0x**. For example, **0x10000**.

-V

Prints debugger version information.

Specification file

The specification file allows you to set up a default set of command line arguments that may be used with any given executable you want to debug. However, not all command line options are available for use in a specification file. If you want a set of default arguments for each program, put the program name at the beginning of a line followed by a space and then a set of command line arguments. The arguments may be continued on the next line if the first

character in that line is a tab. When you run MULTI with the **-m** option, the file listed is checked and if there is an entry that matches the name of the executable being debugged, then that list of command line arguments is used. For example, a specification file named **albatross** might look like this:

```
foo -norc -I /usr/joebob -I /usr/foodir
bar -text 10000 -data 10000
```

If you then type:

```
adamulti -m albatross foo
```

the file **albatross** is searched and the arguments found after **foo** are used. This is equivalent to typing:

```
adamulti foo -norc -I /usr/joebob -I /usr/foodir
```

Resources

To install MULTI, please see the *MULTI 2000 Installation & Licensing Guide*.

The *Debugging with AdaMULTI 2000* manual provides information on using the Debugger and its related features.

The *Quick Reference Card* summarizes the most common Debugger and Editor commands.

For assistance or additional information about the use of Green Hills Software, please contact our Technical Support:

Green Hills Technical Support
North America Mountain/Pacific time, Australia, and New Zealand Tel: (805) 965-6044, Fax: (805) 965-6343 email: support-west@ghs.com
North America Eastern/Central time, South America Tel: (781) 862-2002, Fax: (781) 863-2633 email: support-east@ghs.com
Europe, Africa, India email: support-nl@ghs.com
Japan, Taiwan, and South Korea Tel: +81-3-3576-6805, Fax: +81-3-3576-0106 email: support@adac.co.jp

Chapter 2

Using the Builder

This chapter contains:

- Starting a Builder session
- Setting up your software project
- Navigating through your project
- Setting options: An overview
- Important options
- Building your project
- Debugging

The Builder is a graphical tool that configures how your software project gets built. In addition to maintaining file dependencies like a make file, the Builder also lets you set compiler options.

Starting a Builder session

When you start the Builder, it attempts to use the host system's windowing package. If the Builder is started on a machine with a color monitor, it defaults to color.If you run MULTI from a non-windowing monitor or encounters problems with the window interface, it will come up in non-GUI mode. If MULTI is incorrectly coming up in non-GUI mode, check that the **DISPLAY** environment variable is set, or set it from the command line with the **-display** option.

If you start MULTI with the **-nodisplay** command line option, or if MULTI cannot start in GUI mode, then it will start in non-GUI mode in the Debugger. Because the Builder and Editor are available only in GUI mode, the Debugger is always displayed in non-GUI mode in these scenarios.

To start the Builder from the command line

Whenever you start **adamulti** without specifying any options, the Builder automatically opens **default.bld** in the directory from where MULTI was started. If **default.bld** does not exist in that directory, the Builder opens the most recently used project instead. To start the Builder with a specific build file of your project loaded, do the following:

1. At the command prompt, change to the directory where your project files are located.
2. Assuming that **adamulti** is in your path, enter:

   ```
   adamulti [filename]
   ```

   where *filename* is the build file (***.bld**) of the program, subproject, or library of your project that you want to open.

   Important: If you want to open a build file that is a child of another build file in the hierarchy of your project, use the following syntax to ensure that the child inherits all of the appropriate settings from its parent:

   ```
   adamulti "parent.bld child.bld"
   ```

   where **parent.bld** is the build file from which **child.bld** inherits its option settings.

For example, suppose you want to build the source files that belong to **foochild.bld**, which is a program that inherits options from the top-level program of your project, **masterfoo.bld**. To open **foochild.bld** directly while inheriting all the options needed to compile and link it properly, enter:

```
adamulti "masterfoo.bld foochild.bld"
```

To open a different project in the same Builder window

1. Click Open ().
2. Browse for the ***.bld** of the project you want to open, and click Open.

To open a project in a new Builder window

1. Choose File > Open Project in New Builder...
2. Browse for the ***.bld** of the project you want to open, and click Open.

To set your target

When you start the Builder for the first time, it attempts to select a target that matches the target of your development environment. If the Builder does not select the correct target, you need to manually set it. Once you set the correct target, the Builder will remember and use it every time.

1. Choose Project > Set Build Target for Project...
2. Browse the MULTI installation directory, and select the correct build file for your target. For example, if your target is the PowerPC, select **ppc.bld**.

Note: Your MULTI installation may contain more than one build file for your target. The majority of users select the most basic build file, for example, **ppc.bld**. If you have a customized MULTI environment, you may need to load a specialized build file for your target. For example, if you are using the PowerPC 500 as your target, load **ppc500.bld**.

Setting up your software project

By creating a hierarchical view of all your programs, libraries, source files, headers, and other project files, you can define the file dependencies of your software project.

As you define your file dependencies, you need to add a build file (***.bld**) for every program in your project. Also, add a build file for libraries that you want

to rebuild with your project. If you would like to take this a step further and group source files into modules, you may add build files for your subprojects. These build files:

- List all of the files that comprise the program, subproject, or library.
- Store compiler options for the program, subproject, or library, and for each source file.

When defining your project hierarchy, be aware that files inherit compiler options from parent build files. For example, suppose a program, **masterfoo.bld**, contains a subproject, **subfoo.bld**. In this scenario, **subfoo.bld** is the child that inherits options from the parent, **masterfoo.bld**. Now suppose that **subfoo.bld** contains a source file, **foo.c**. In this scenario, **foo.c** inherits options from **masterfoo.bld** AND **subfoo.bld**.

Using default.bld

When you start creating your project hierarchy, the Builder automatically puts **default.bld** at the top of the hierarchy. Since your entire project inherits from **default.bld**, you can use **default.bld** to set global options that will be used as the default throughout your project.

You can create your own top-level ***.bld** files directly below **default.bld** in the hierarchy if you are uncomfortable with using **default.bld** at the top of your hierarchy. You can then load one of your own ***.bld** files when you start the Builder. Your project will still inherit from **default.bld** behind the scenes, but if you never load **default.bld**, the relationship will have no impact.

To define the executable programs in your project

You define what programs get built in your project by adding a build file for each program. You can have multiple programs that are built as part of your project. These programs can be built at the same hierarchical level, or one program can be a child of another program.

If a program is a child of another program (i.e., it is lower in the hierarchy), the source files of the child will inherit default settings from the parent. However, the child's source files will be compiled and linked into a separate executable from the parent executable. For example, suppose you have a program build file, **masterfoo.bld**, at the top of your project's hierarchy. Lower in the hierarchy, you have **fooaid.bld**, the build file for a separate program. The source files contained in **fooaid.bld** inherit options from **masterfoo.bld**, but are compiled and linked into a separate executable, **fooaid.out**.

1. Start the Builder from the project's directory.

2. If you want to put the program at the top of your project's hierarchy, make sure **default.bld** is at the top of the Source pane.
   – or –
   If you want to put the program lower in your project's hierarchy, navigate until the program or subproject that will contain the new program is at the top of the Source pane.
3. Click the Add button ().
4. In the file chooser, browse for a location to create your new build file. It is recommended to place the program's build file in the same directory as the source files for that program. When you have chosen a directory, type in the name of the build file to create. Usually, this will be the same name as the executable you wish to generate, but with a **.bld** extension. For example, if your executable is called **masterfoo.out**, call your build file **masterfoo.bld**.
5. Click Add.

To define a subproject

To organize your source code into logical units, use subprojects. A subproject is always a child project of a program or another subproject. The source files contained within a subproject are compiled and linked into its parent program; subprojects do not get built into executables. You define a subproject as a special type of build file (***.bld**) that does not get built.

If your project uses sub-directories to organize your source code in your file system, you can create the subproject's build file in the appropriate sub-directory. This allows the Builder to look in the correct directory when you add source files to the subproject.

1. Double-click the build file of the program or subproject that will contain the new subproject.
2. Click Add ().
3. In the file chooser, select a directory and enter the name of the build file for the new subproject. The subproject's build file needs to have a **.bld** extension. Normally, the subproject is located in the same directory as the source files contained within it.
4. Click Add.
5. Highlight the new subproject.
6. Choose Project > Options for Selected Files....
7. On the General tab, set the Type: field to Subproject.

8. Click OK.

To link in a compiled library

To link a compiled library to a program, add the library file to the program's build file.

1. Double-click the build file of the program to which you want to link the library.
2. Click Add ().
3. Browse for the library file that you want to add, then click Add.
4. Look at the new file's type to make sure the Builder assigned it the type Library. If the Builder did not assign the Library type:
   a. Highlight the library file.
   b. Choose Project > Options for Selected Files... .
   c. In the General tab, set the Type text field to Library.

To link to a library that gets built with your project

You can define a library and its source files so that the library gets rebuilt every time you start a build.

If the library is hierarchically lower than a program or subproject, the library will inherit settings from the parent program or subproject. However, the library's source files get built into the library, not into the parent program.

1. Double-click the build file that will contain the new library.
2. Click Add ().
3. In the file chooser, select a directory and enter the name of the build file for the new library. Make sure your library's build file has a **.bld** extension. Normally, your library's build file is located in the same directory as the library's source file, and the name of the build file is the same name as the library, but with a **.bld** extension.
4. Click Add.
5. Highlight the new library.
6. Choose Project > Options for Selected Files... .
7. In the General tab, set the Type text field to Library.
8. Double-click the new library.

9. Add source files for the new library.

To add an existing source file to your project

1. Double-click the build file that will contain the source file.
2. Click Add ().
3. Browse for the source file, and click Add. To add multiple files quickly, you may specify a file pattern using the wildcard characters '*' and '?' in the file chooser.

To define and create a new source file

1. Double-click the build file that will contain the new source file.
2. Click Add ().
3. In the file chooser, select a directory and enter the name of the new source file. Click Add. The name of the file appears in the Source pane, but the actual file does not yet exist.
4. Double-click the name of the new source file to open it in the Editor.
5. Save the new file.

To define header files

You can add header files to your project hierarchy to quickly access the header files for edit. However, adding a header file to the hierarchy does NOT ensure that the Builder will find the header when it is compiling your source files.

To define a header file in your project hierarchy, add the file the same way as adding a source file.

To ensure that the Builder finds the header file when it is compiling your source files:

1. Select the build file for the program or library you are building.
2. Choose Project > Options for Selected Files..., then navigate to the General tab.
3. In the Source Directories: field, enter the path where your header files are located. If your header files are located in multiple directories, separate the paths by commas WITHOUT any spaces.

To change a file's type

The Builder automatically sets the type of file based on the file's extension. If you need to manually override the Builder's choice:

1. Select the file.
2. Choose Project > Options for Selected Files... .
3. Go to the General tab.
4. In the Type: field, select the type of the selected file.

Note, however, that the compiler will always be selected based on the file's extension, not the file type. So for instance, even if you set the type for foo.c to Fortran, the C compiler will still be used instead of the Fortran compiler. Therefore, you will need to rename the file if you want to use a different compiler.

To rearrange the order of files in the hierarchy

The files at each level of your project's hierarchy display in the order in which you added them to the Builder. This is also the same order in which the files will be built. You can rearrange the order in which the files are displayed to change the build order, and also to help you visually understand your hierarchy. For example, you may want to rearrange the files so the source files that belong to a build file appear before the subprojects of that build file.

1. Highlight the files you want to move.
2. Choose Edit > Cut Selected Files, or click the Cut button ().
3. Highlight the file immediately above where you want the files to be placed.
4. Choose Edit > Paste Selected Files, or click the Paste button ().

Alternatively, after you highlight the files you want to move, press Ctrl+UpArrow or Ctrl+DownArrow to move the files up or down in the list.

Navigating through your project

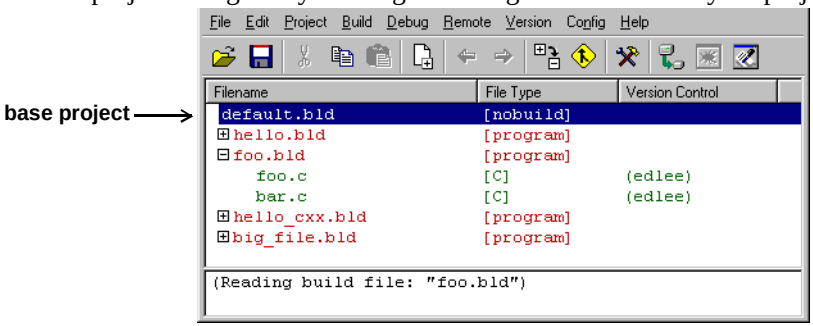
In the Builder's hierarchical view of your project, files that belong to a build file (program, subproject, or library) appear below the build file, and are indented to the right.

There are two ways to see the contents of a build file in the hierarchy:

- Click the plus (+) and minus (–) signs to expand or collapse the contents of a program, subproject, or library. This does not change the base project.
- Double-click a program, subproject, or library to make it the new base project. At this point, you can no longer see the parent build file that contains the new base project.

The base project

The build file that appears at the top of the Source pane is the base project. The base project changes as you navigate among the build files in your project.



Knowing which build file is the base project is important for three reasons:

- When files are added, they are added to the base project.
- Options for build files get set differently depending on whether the build file is the base project. For more details, see “Setting options for programs, subprojects, and libraries” on page 21.
- When you start a standard build, the current base project is what gets built, regardless of what is highlighted in the Source Window.

  For example, suppose masterfoo.bld is the current base project, and contains another program, foochild.bld. If you highlight foochild.bld and then choose Build > Build Program, the Builder compiles the masterfoo.bld program, NOT foochild.bld. To build foochild.bld, you must double-click foochild.bld to make it the base project, then start the build.

Navigating among base projects

During a Builder session, you need to navigate through the various build files of your project hierarchy. To descend one level in the project hierarchy, double-click on the build files to navigate deeper. Then, you can quickly

navigate back and forth between levels by using the Back and Forward history buttons:

 Navigates up the project hierarchy to the build file that encloses the current base project.

 Navigates down the project hierarchy to the location of the previous base project opened.

Searching through your project

If you have a big project, it may be difficult to locate a particular file within the hierarchy. You can perform an incremental search to find your files quickly. To perform the search, first expand the projects that contain the files you are searching for. To expand the projects, click on the plus signs or click the Expand button (). Then, use one of the following keystrokes:

Ctrl+f searches forward for the *filename*.

Ctrl+b searches backward for the *filename*.

When you type a key combination, the first instance of the matching *filename* is highlighted. To find the next match, repeat the key combination. When you are done, press Escape to cancel the search mode.

Note that only files in projects which have been expanded will be found. This is why it is important to expand the relevant projects first, by clicking on the plus signs, or using the Expand button.

To view all files in the base project

To expand the hierarchical view of the current program, subproject, or library so you can view all of its files, click ().

To open a source file in the Editor

If you want to edit a source file that appears in the hierarchical view of your project, simply double-click the filename.

Setting options: An overview

You can use the Builder to set options for your project such as how it gets compiled and linked, how you debug the executable, and how runtime error checking works. Many of these options correspond directly to the compile-time options for the Green Hills compilers.

All of the Builder options are available through menu items in the Build menu. For detailed descriptions of each Build menu item, see “Build menu” on page 37.

Inheriting options from parent build files

A file inherits its options from all build files (***.bld**) to which it belongs in the project hierarchy. You can override these inherited options by setting options in the file itself. When the Builder compiles your project, the top-level project’s options are applied first, with each subsequent child project adding its options to the existing ones, overriding or appending where specified, until finally the individual file options are added for each file.

Viewing inherited options

When you look at the options for a particular file, whether it is a source file or build file, you see only the options that have been specifically set for that file. You do not see the options that the file inherits from the build files of its parents. Before setting an option for a specific file, you probably want to see whether that option is already being inherited from a parent.

To see the inherited value of the option, do one of the following:

- In the main Builder, click Merge ().
- From the Builder, choose Project > Options for *Selected Files*... and click Merge in the Options dialog box.

You cannot edit the options while you are looking at them in merged mode. If you see an option you want to change, first do one of the following:

- In the main Builder, click Unmerge ().
- From the Builder, choose Project > Options for *Selected Files*... and click Unmerge in the Options dialog box.

Then, edit the option.

Setting options for programs, subprojects, and libraries

Because you can reuse build files for programs, subprojects, and libraries in multiple projects, it is important to specify whether the options for a build file are specific to the current project or whether the options are specific to the build file itself regardless of what project it belongs to. This choice, which is based on which build file is the base project, determines if the options are stored in the

build file itself or in its parent's build file. For more information about the Builder's base project, see "Navigating through your project" on page 18.

- If the options are specific to the current project, set the options while the parent's build file is the base project. For example, suppose foochild.bld is a subproject of masterfoo.bld. Make sure masterfoo.bld appears at the top of the Source Window, then select foochild.bld and set the options.

  Options set in this way are stored in the parent's build file (in this example, masterfoo.bld). If you reuse the child's build file in a different project, the options will not be carried over into the new project.

- If the options are specific to the build file itself, set options for the build file while it is the base project. For example, suppose foochild.bld is a subproject of masterfoo.bld. Make sure foochild.bld is at the top of the Source Window before you select foochild.bld and set the options. You will not see masterfoo.bld in the Source Window.

  Options set in this way are stored as part of the child's build file (in this example, foochild.bld), and therefore will be set for that build file regardless of what parent it belongs to.

Setting options for source files

You can set options for an individual file or for a list of files.

- To set an option for an individual file, select the file in the Source Window and set the options. Be aware that if you move the file to a different program, subproject, or library, the options that you previously set do NOT move with the file.
- To set an option for a group of files, select the build file that contains the files in the Builder's hierarchy of your project, then set the options. The options are set in the build file; all files below the build file, including its source files, will inherit these options.

Understanding tick boxes

Tick boxes allow you to set the value of some options. These tick boxes have three states:

 On. The option is turned on regardless of the default setting inherited from a parent build file.

 Off. The option is turned off regardless of the default setting inherited from a parent build file.

Default. The value of this option is inherited from the parent build file. To see the inherited value of the option, click Merge in the Project > Options for *Selected Files*... dialog box, or click Merge ( ) in the main Builder window.

Entering multiple text items for an option

Some options require that you enter a text item, such as a path or *filename*. If you need to enter multiple items for these options, then separate each item by a comma, without any spaces. Do not put spaces before or after the commas because the Builder will not understand your items.

Important options

To set optimization options

To set compiler optimization options that correspond to the optimizations options found in the Green Hills Language User's Guide for your programming language, choose Project > Options for *Selected Files*... > Optimization tab. If you have questions about individual options, see "File Options > Optimization tab" on page 52.

To set run-time error checking options

To enable the Debugger to provide run-time error checking, set run-time error checking options in your project before you build it. The options that you set determine the types of errors the Debugger will be able to check at run-time.

To set run-time error checking options, choose Project > Options for *Selected Files*... > Run-time Error tab. If you have questions about individual options, see "File Options > Run-time Error tab" on page 55.

To set manifest constant definitions for the preprocessor

1. Navigate to the top of your hierarchy to ensure that the constant definition is inherited throughout your project.
2. Choose Project > Options for *Selected Files*... .
3. Choose the General tab.
4. In the Defines: field, enter the constant definitions. Do not enter the **-D** preprocessor option before the constants. For example, if you want to set DEBUG to 1 and MAX to 64, enter:

```
DEBUG,MAX=64
```

See “Defines:” on page 50, for more information.

To undefine manifest constant definitions for the preprocessor

1. Navigate to the top of your hierarchy to ensure that the undefined constant definition is inherited thought your project.
2. Choose Project > Options for Selected Files... .
3. Select the General tab.
4. In the Undefines: field, enter the constant definitions you want to undefine. Do not enter the **-U** preprocessor option before the constants.

Building your project

To start a build, make the program’s build file the base project, then choose Build > Build *program-nam*e or *library-name* or click (). When you start a build in this way, the Builder completes the build according to the options set in the Build Panel.

To change options in the Build Panel, choose Build > Advanced Build Controls... . For information about an individual build option, see “Build Panel” on page 43.

To perform a dryrun build

You can run a build that duplicates the command line options **-dryrun** and **-#**. The build will show the actions that take place without actually performing them.

1. Choose Build > Advanced Build Controls... .
2. Choose Test Run.
3. Choose Commands.
4. Click the Build button ().

To build individual source files

1. Select the source files that you want to build.
2. Choose Build > Build Selected Files.

To specify the name of the compiled program or library file

By default, the Builder names the compiled program or library file based on the *.bld file that gets built. To override the default name:

1. Navigate to the *.bld that you want to change.
2. Choose Project > Options for Selected Files... .
3. Choose the Actions tab.
4. In Output Filename, enter the name you want for your program or library.

To track down errors from a build

1. Build your application.
2. In the Progress Window, double-click the error message you want to examine.

   The Editor appears, opens the source file that contains the error, and puts the cursor on the line of code that caused the error.

Building platform-specific programs from the same source files

When a program needs to work on multiple hardware platforms, often times the source files are identical except for one or more assembly language routines that vary from processor to processor. In cases like this, you can create Select One subprojects that contain the processor-specific files so you can build the same program for multiple target processors.

Consider the following example. Suppose a program for SH processors depends on the following files:

```
indprogram.c
indprogram.h
traps.ppc          ; PowerPC assembly language file
```

Suppose the same program compiled for Alpha depends on these files:

```
indprogram.c
indprogram.h
traps.mip          ; MIPS assembly language file
```

You can use a Select One subproject, **traps.bld**, to build a program that uses only the processor-specific file, **traps.ppc** or **traps.mip,** for the specified target processor.

If you have multiple files that are specific to a single processor, create multiple Select One subprojects. Suppose the program in the above example also uses

bdriver.ppc and **bdriver.mip**. In this case, you would need another Select One subproject, **bdriver.bld**.

To define your project for multiple platforms

1. Create the program that gets compiled and linked.
2. Add to the program all of the source files except the processor-specific assembly files.
3. Add to the program a Select One subproject that contains the processor-specific assembly files:
    a. Create a build file for the subproject. For example, create traps.bld.
    b. With the new build file highlighted, choose Project > Options for Selected Files... .
    c. Choose the General tab.
    d. In the Type: field, select Select One.
    e. In the Source Window, add to this new Select One build file the various processor-specific assembly files for the program. In the example above, traps.bld contains two files: traps.sh and traps.alp

To build a platform-specific program

Once you have defined your project for multiple platforms using a Select One subproject:

1. Set the Builder's Target: field to specify the target processor for the current build. For details about setting the target, see "To set your target" on page 13.
2. With the program as the base project and highlighted, choose Project > Options for Selected Files... .
3. Choose the Configuration tab.
4. In the Select text field, enter one or more of the file extensions needed when building for this particular target. That is, enter the extensions of the files that apply specifically to the target processor. In the example above, if you are building your program for PowerPC, enter **ppc** in the Select text field. If you are building your program for Mips, enter **mip** in the Select text field.
5. Start the build.

When you start the build and each time the Builder reaches a Select One subproject, it chooses the first file in the list that matches one of the extensions

specified in the Select: field of the program. The other files in the Select One subproject are ignored.

Debugging

You can use the Builder to set options that determine the type of debugging information that gets generated when you build your project. Once you build your project, you can use the Builder to connect to a simulator or debug server and to start a debugging session on your new executable.

To set what debugging information gets generated

The default is to have MULTI level debugging information generated.

1. Choose Project > Options for Selected Files... .
2. Select the General tab.
3. Set the Debugging Level: field to the desired value.

To connect to a target through a debug server or simulator

Before you can debug your application on your target system, you must first connect to a remote target through a debug server or simulator.

1. Click Connect () or choose Remote > Connect to Target.
2. In the dialog box that appears, enter the debug server or simulator that you are using in your debugging environment.

The available debug servers include:

MULTI debug servers
810serv, 830serv, 850serv
960serv
e7kserv
hpserv
idtserv
m16serv
monserv
ocdserv
rtserv
spotlight (for pSOS)
tornserv
unixserv
vxserv
winserv

The available simulators include:

MULTI simulators
sim800, sim850
sim960
simalp
simarm
simm16
simmma
simmips
simppc
simrh32
simsh
simspc

3. You may also add command line options following the name of the debug server. For example: **simppc -ppc860**
4. Click Ok.

When you connect to a debug server or simulator, an IN/OUT window and a Target window will appear. You can close these windows without harming the MULTI environment. If you close these windows and later wish to use them again, you can choose Remote > Show Target Windows to redisplay them.

The IN/OUT window provides the basic I/O for the program you are debugging. The Target window allows you to send commands to the debug server or simulator.

You can perform incremental searches in both the IN/OUT and the target window. The key presses are:

Ctrl+f Searches forward.

Ctrl+b Searches backward.

To start a debug session

1. Connect to your debug server or simulator. Alternatively, you can wait to connect to the debug server or simulator from the Debugger.
2. Double-click the project that you wish to debug.
3. Click Debug () or select Debug > Debug *program-name*.
4. If the user entry point is known, then the Debugger displays it on start-up. Otherwise, the Debugger displays the executable entry point on start-up. For normal C applications, the user entry point is **main** and the executable entry point is usually **_start**.

Chapter 3

The Builder GUI

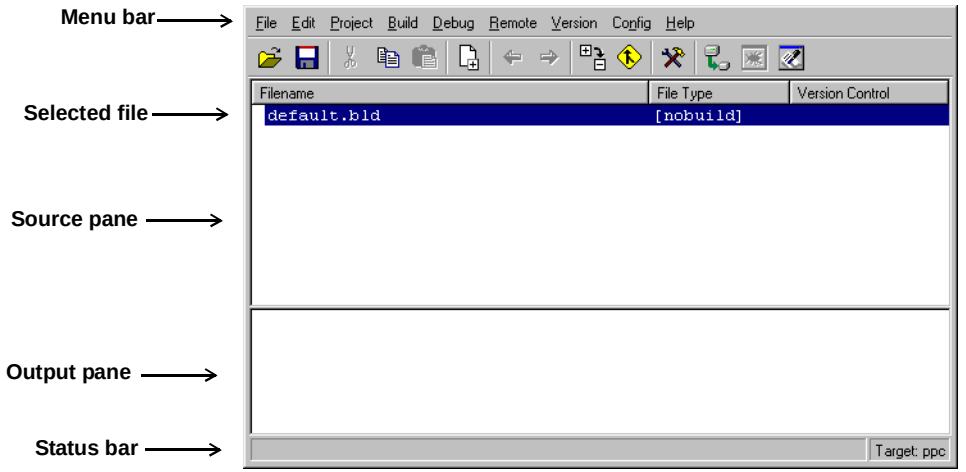
This chapter contains:

- The Builder window
- The Builder menus
- The Builder toolbar
- Other Builder components
- Build Panel
- File Options dialog box
- Language Options dialog box
- CPU Options dialog box
- Toolchain Options dialog box
- The Progress window

This chapter provides a comprehensive description of the commands and options in the main Builder window menu bar.

The Builder window

To start MULTI and open the Builder windows, enter **adamulti** at the command line, if MULTI is in your path.



Title bar

The title bar contains the title of the current project. In this example, the project title is **default.bld**, the default project title when you first run MULTI.

The Builder menus

NOTE: The following tables contain all Builder menu items along with brief descriptions of the items. If a menu item has a command equivalent, then it is provided for advanced users wishing to configure their Builder menu settings. To configure Builder menu settings, choose Config > Options... > General tab and click the Menus... button. A dialog box appears where you can enter the command equivalents. See Chapter 10, “Configuration commands” for more information.

Pop-up menu

Many command operations can be performed in the builder by right-clicking in the source pane. All operations available on the pop-up menu are documented elsewhere in this chapter, and apply to the currently selected files.

File menu

NOTE: Spaces are not allowed in filenames. This restriction applies throughout the entire MULTI development environment.

File menu (builder)		
Menu item	**Meaning**	**Command**
Open Project in Builder...	Opens a project (*.bld) in the current Builder window.	Open
Open Project in New Builder...	Opens a project (*.bld) in a new Builder window.	NewBuilder
Open File in Editor...	Opens a file in a new editor window.	Edit
Save *project*	Saves changes for all projects in the hierarchy that have been modified.	Save
Save *project* As...	Same as **Save**, except you can save the current project under a new name.	SaveAs
Revert *project*	Reloads the current project from disk, discarding any changes made since the last save.	Revert
Print Current View...	Prints the currently displayed project hierarchy. The hierarchy will be printed in the exact state that it is displayed in the source pane. In particular, if you wish to print the contents of subprojects, you need to expand them first so that they are displayed.	Print
Print Entire Project...	Prints the fully expanded project hierarchy. All projects will be expanded first, then the entire project hierarchy will be printed.	PrintEntire
Write entire project to file...	Prints the currently displayed project hierarchy to a text file in ASCII format.	PrintToFile
Recent Files	This submenu contains recently edited files. You can choose one of these files to edit it more quickly.	Edit
Recent Projects	This submenu contains recently opened projects. You can choose one of these projects to open it more quickly.	Open
Close Builder	Closes the Builder window.	Close
Exit All	Exits MULTI. If any files are not saved, a **Save All** dialog box appears first. All active debug sessions are terminated.	QuitAll

Edit menu

Edit menu (builder)		
Menu item	**Meaning**	**Command**
Cut *Selected Files*	Removes the selected files from the current project and places them in the clipboard. **Note:** For the commands Cut Files, Copy Files, and Paste Files, an internal clipboard separate from the system clipboard is used.	CutFiles
Copy *Selected Files*	Makes a copy of the selected files and places them in the clipboard.	CopyFiles
Paste *Selected Files*	Copies files from clipboard and inserts them into the top level project, after the current selection.	PasteFiles
Find in Files...	Greps through the files in your project.	Grep

Project menu

Project menu (builder)		
Menu item	**Meaning**	**Command**
Add Files To *Project*	Opens a dialog box to add selected files to the current project.	Add
Remove *Selected Files*	Removes the currently selected files from the project.	Remove
Edit *Selected Files*...	Opens in the Editor the files that are selected in the source pane.	EditSelected
Simplify Filenames	Attempts to convert absolute filenames to relative filenames when possible. To be more precise, if the path of a file is removed and if the file can still be found by searching the source directories list, then the full pathname is replaced by the filename without a path. This is a simple means of converting absolute pathnames in projects to short relative pathnames to increase portability. To add source directories, select the project first, then select Project > Options for *Selected Files*... > General tab.	SimplifyNames

Project menu (builder)		
Menu item	**Meaning**	**Command**
Recalculate Filenames	For efficiency, the Builder only evaluates filenames against the “source directories” when a project is opened. To add source directories, select the project first, then select Project > Options for *Selected Files*... > General tab. It continues to use the filename calculated initially unless Recalculate Filenames is selected. Choosing Recalculate Filenames rescans the source directories list for the files listed in the current project. This is useful after changing the source directories list, where a file originally found in one directory is now in another. It is also useful if a file is moved from one source directory to another, or a file is deleted from a source directory. Without performing this operation, the Builder would continue to use the file in the original directory.	RecalculateFilenames
Set Build Target for *Project*...	Allows you to select a new target system for which to build. All files in a project inherit a set of default options that depend upon the target system. These defaults are set up in a build file which consists of the target name followed by the **.bld** extension, such as **ppc.bld**. Some targets are preinstalled in the same directory in which MULTI was installed. You may create your own target files as well.	n/a
Options for *Selected Files*...	Displays the File Options dialog box, which you use to set most build options. For more details, see “File Options dialog box” on page 45. **Note:** Only the options for the first file selected in the source pane will be affected.	Options FileOptions

Project menu (builder)		
Menu item	**Meaning**	**Command**
Language Options for *Selected Files*...	Displays the Language Options dialog box, which you use to set language specific compiling options. These include Ada, C, C++, FORTRAN, and Pascal options. See "Language Options dialog box" on page 68 for more information. **Note:** Only the options for the first file selected in the source pane will be affected.	Options LanguageOptions
CPU Options for *Selected Files*...	Displays the Options dialog box that corresponds to the processor family for which you are building your program. The processor family is determined by the target you have selected (see "Set Build Target for Project..." on page 36 for more information). For example, if you are building your program for the PowerPC, the PowerPC Options dialog box appears. See "CPU Options dialog box" on page 92 for more information on choosing targets and a list of possible CPU Options dialog boxes. **Note:** Only the options for the first file selected in the source pane will be affected.	Options Cpu
Toolchain Options for *Selected Files*...	Displays the Toolchain Options dialog box that corresponds to the toolchain for which you are building your program. This dialog box includes linker options and assembler options. The toolchain is determined by the target you have selected (see "Set Build Target for Project..." on page 36 for more information). See "Toolchain Options dialog box" on page 118 for more information on the Toolchain. **Note:** Only the options for the first file selected in the source pane will be affected.	Options Toolchain

Build menu

Build menu (builder)		
Menu item	**Meaning**	**Command**
Build *Program*	Builds the current program and shows the status of the build in a separate window. For any type of build, the project will be saved first if it has been modified.	Build
Build *Selected Files*	Builds only the selected files instead of the whole project.	Build Selected
Rebuild All	Builds the current project, and forces every file to be rebuilt, even if the dependencies show that the file has already been built and is up to date.	Build All
Build and IgnoreErrors	Builds the current project, and continues building upon detection of an error. Normally, the build stops when an error occurs.	Build IgnoreErrors

Build menu (builder)		
Menu item	**Meaning**	**Command**
Cleanup Intermediate Files	Deletes all of the files which are normally created when building the project. This includes object files, libraries, and executables. In other words, at each step where a file would be created in a normal build, the file is deleted instead. The only files that remain after **Clean Up** are the source files necessary for building the project from scratch.	Build CleanUp
Show a Dry-run of Build	The Builder determines and displays the steps of building the project, but does not actually run any of the tools, such as the compiler, assembler, linker, archiver, etc. This option is usually used in conjunction with one of the Display Overrides settings in the Build Panel. In particular, Progress, Reasons, and Commands can all be displayed in a test run. The Warnings setting has little effect in Test Run mode because only the builder itself is executed in Test Run mode, therefore only warnings from the builder itself are displayed. Enabling both **Test Run** and **Commands** is equivalent to the **-dryrun** and **-#** build-time options on the driver command line.	Build TestRun
Advanced Build Controls...	Opens the Build Panel dialog box, which you use to temporarily set options for how you want to build the project. These settings will be in effect for the current session only, and will not be saved when you close the Builder. See “Build Panel” on page 43 for more information on each build panel options.	BuildPanel

Debug menu

Debug menu (builder)		
Menu item	**Description**	**Command**
Debug *Program* ...	Starts a debugging session on the compiled executable of the current project.	DebugCurrent
Debug Other...	Opens a dialog box which you can use to select the executable that you want to debug.	Debug
Attach to Process...	Attaches to a running process. See also the debugger command **attach**. This command works only with a multi-tasking target and is grayed out otherwise. It opens a new debugger window to debug the specified task.	n/a
1,2,3,4	Debug the specified executable.	Debug

Remote menu

Remote menu (builder)		
Menu item	**Description**	**Command**
Connect to Target...	Opens a dialog box where you can enter the debug server or simulator to which you want to connect. The four most recently connected targets are available in the drop-down list.	Remote
Disconnect from Target	Disconnects from the remote target.	Disconnect
Show Target Windows	Displays the Target and I/O windows for the target to which you are currently connected.	TargetWin
Load Module	This submenu is for multi-tasking debug servers only, allowing you to download a new object module to the target. Choose Load Module... again from the submenu to choose the module to download from a dialog box, or choose one of the recently downloaded projects from the list provided.	LoadModule
1,2,3,4	Connect to the specified remote target.	Remote

Version menu

Version menu (builder)		
Menu item	**Description**	**Command**
Check Out	Retrieves a writable copy of the latest version and locks the file so others cannot change the file while you work on it.	BCheckOut
Check In	Saves the changes made to the file, makes the file read-only, and removes the lock from the file. You will be asked for comments to be saved in the log file along with your changes.	BCheckIn
Check In + Out	Saves the changes made to the file, but keeps the file checked out (locked).	BCheckInOut
Retrieve	Retrieves a read-only copy of the current version of the file, even if the file is locked.	BRetrieve
Discard Changes	Discards changes made to the file, removes the lock from the file, and reverts to the latest version. Use this to undo a checkout when you decide not to make any changes.	BUncheck

Version menu (builder)		
Menu item	**Description**	**Command**
Place Under VC	Puts the current file under version control. Once a file is placed under version control, the file must be checked out before changes can be made.	BCreate
ShowHistory...	Opens a window that displays version history information: version numbers, dates, user names, and comments.	BShowHistory
Other VC Command	Allows you to run additional version control commands. For a complete list of the version control commands, see Chapter 4, “Version control”.	BOther

Config menu

Config menu (builder)		
Menu item	**Meaning**	**Command**
Options	Displays the Options dialog box, which you use to change options that affect the way the Builder and other MULTI tools look and behave.	ConfigOptions
Save Configuration as Default	Allows you to permanently save the changes you made in the Options dialog box.	SaveConfig
Clear Default Configuration...	Clears all saved changes and reverts to all defaults in the Options dialog box.	ClearConfig
Save Configuration...	Save the changes you made in the Options dialog box to a user specified file.	SaveConfigToFile
Load Configuration	Load changes to the settings in the Options dialog box from a user specified file.	LoadConfigFromFile

See Chapter 10, “Configuration commands” for more information.

Help menu

Help menu (builder)		
Menu item	**Meaning**	**Command**
Builder Help...	Opens MULTI's online help for the Builder.	Help
Manuals	Opens the "Manuals sub-menu", which will display a list of manuals appropriate to your version of MULTI. Choosing one of these manuals will open the online help to the first page of that manual.	n/a
About MULTI...	Displays the About banner.	About

The Builder toolbar

These are the buttons on the Builder toolbar, their meanings, and their equivalent commands:

Builder toolbar		
Button	**Meaning**	**Command**
	Opens a project (*.bld) in the current Builder window.	Open
	Saves changes for all projects in the order that they have been modified.	Save
	Cuts the selected files from the current project and places them in the clipboard. For the commands Cut Files, Copy Files, and Paste Files, an external clipboard separate from the system clipboard is used.	CutFiles
	Makes a copy of the selected files and places them in the clipboard.	CopyFiles
	Copies files from clipboard and inserts them into the current project.	PasteFiles
	Opens a dialog box to add selected files to the current project.	Add
	Click to navigate up the project hierarchy to the build file that encloses the current build file.	Back
	Click to navigate down the project hierarchy to the previous build file that you were working with.	Forward

Builder toolbar		
Button	**Meaning**	**Command**
 Expand  Contract	Click Expand to display an expanded view of all projects. In other words, each project will be expanded (have its plus sign clicked). Click Contract to display a contracted view of all projects. Each project will be contracted (have its minus sign clicked). Click Expand to display expanded view of subprojects. Click Contract to hide the contents of all subprojects.	ExpandAll
 Merge  Unmerge	Merge and Unmerge. Clicking Merge shows the merged options for a file, including full pathnames and all inherited options. This will affect the Builder window and all of its options windows as well. After clicking this button, it becomes Unmerge, which changes the options view back to normal. See “Viewing inherited options” on page 21 for more information.	ShowMerged
	Saves a project, if necessary, then builds it. The status of the build is shown in a separate window.	Build
 Connect	Connect to a debug server or simulator.	Remote
 Disconnect	Disconnect from a debug server or simulator.	Disconnect
 Debug	Debug the current program.	DebugCurrent
 Edit	Edit the currently selected files.	EditSelected
	Closes the Builder window. You will be prompted to save and/or check in all edited files. You can configure whether or not to have this button on the toolbar. See also “Display close (x) buttons” on page 243.	Close

Other Builder components

Source pane

Your projects and source files are listed in the source pane. Each line of this list is displayed with the filename in the Filename column, followed by the file type in square brackets in the File Type column. If you are using MULTI Version Control (MVC), then files that are checked out have the appropriate user name in parentheses in the Version Control column. All projects (*.bld) have a small plus or minus to their left. Clicking the plus will expand the view so that the contents of the project are displayed (the plus will change into a minus).

Clicking the minus will contract the view so that the contents of the project are hidden (the minus will change into a plus).

When the names of files in a project do not show an absolute path, the path is relative to the directory the build file resides in. Your project is more portable if you use only relative file names. You can specify additional source directories in which to find files by doing the following: select the project you wish to add source directories to, then choose Project > Options for *Selected Files*... > General tab, and edit the Source Directories list. After you add source directories, choose Project > Simplify Filenames, which will attempt to convert absolute filenames to relative filenames.

You can change the order of the files in the list by selecting a file in the list, and then while holding the Ctrl key press the up or down arrow keys to move the file up or down.

Output pane

The Output pane displays information about the current status of the build project.

Status bar

When the cursor is on a button, the status bar shows a short description of the function of the button.

Target window

The target window displays the currently selected build target. To change the build target, select Project > Set Build Target for Project....

Build Panel

(Builder : Build > Advanced Build Controls....)

This sets options for how you want to build your project.

The **Build Options** check boxes affect the way the files are built.

Build all

Forces every file to be rebuilt, even if the dependencies show that the file has already been built and is up to date.

Ignore errors

Continues building upon detection of an error. Normally, the build stops when an error occurs.

Clean up

Deletes all of the files which are normally created when building the project. This includes object files, libraries, debugger symbol files, and executables. At each step where a file would be created in a normal build, the file is deleted instead. The only files that remain after **Clean up** are the source files necessary for building the project from scratch.

Test run

The Builder goes through the steps of building the project, but does not actually run any of the tools, such as the compiler, assembler, linker, archiver, etc. This option is usually used in conjunction with one of the **Display Overrides** settings described below. In particular, **Progress**, **Reasons**, and **Commands** can all be displayed in a test run. The **Warnings** setting has little effect in **Test run** mode because only the builder itself is executed in **Test run** mode, therefore only warnings from the builder itself are displayed.

Enabling both **Test run** and **Commands** is equivalent to the **-dryrun** and **-#** build-time options on the driver command line.

The **Display Overrides** check boxes affect the appearance of the build's output. These check boxes override display settings, which can also be set for your project in Project > Options for *Selected Files*... > General tab > Show. When the check box displays a plus (), the option is turned on regardless of the setting in the project. When the check box displays a minus (), the option is turned off regardless of the setting in the project. When the check box displays nothing (), the setting in the project is used.

Progress

Displays the build steps as they occur. The default setting is on.

Warnings

Displays any warnings that occur from the compilation process. The default setting is on.

Reasons

Displays why a certain action takes place. The default setting is off.

Commands

Displays the actual program and arguments used in the step. Together with **Test run**, this is equivalent to the **-dryrun** and **-#** build-time command line options. The default setting is off.

File Options dialog box

(Builder : Project > Options for *Specified Files*....)

Each of the Options menu items in the Project menu opens a dialog box that allows you to set certain options. The File Options dialog box contains several tabs. Each tab contains options that affect your project and how it gets built. Most of the options are compile-time options described in the Green Hills *Language User's Guides* and the *Development Guides*. For an overview of issues involved when setting options, see "Setting options: An overview" on page 20.

Note: The File, Language, CPU, and Toolchain options dialog boxes ALL have the following buttons:

Merge

Allows you to see the inherited settings for options which have not been set. When the dialog is displaying merged options, the settings may not be modified. See "Viewing inherited options" on page 21 for more information.

OK

Applies any changes which have been made in the dialog box and closes it.

Cancel

Discards any changes and closes the dialog box. Certain operations, such as Merging or selecting another file in the Source Window, require first that you apply or discard any changes. You will then be prompted to choose whether to apply or discard your changes for each Options dialog box which has been modified.

Apply

Applies any changes and leaves the dialog box open.

File Options > General tab

(Builder: Project > Options for *Selected Files*... > General tab)

Type (drop-down list)

The Type drop-down list box allows you to select a file type. The Builder normally tries to determine the file type by looking at the file's extension. To change the file type, open the Type box, then choose the desired type from the drop down list. For example, to change a build file from a program to a subproject, click the text program and select Subproject. Then click Apply. The build file will show the newly assigned type in the build file list. MULTI does not allow changing a file type to certain inappropriate file types. Below is an explanation of each file type:

Type (drop-down list)	
File Type	**Meaning**
Default	Determines file type by the file extension.
Nobuild	This type of build file is useful for containing other projects, such as programs and libraries. Building a project of this type will build all the projects contained within it. The file **default.bld** is normally set to this type, since it contains all the projects you will want to build. If a nobuild is contained within another build file (for example a program), then building the program will not build the contents of the nobuild. Instead, navigate into the nobuild and build it directly from there.
Program	Main build file for a project; generally contains source files and/or subprojects. The contents of this project are compiled and linked to form a program.
Subproject	A component of a project. Subprojects are useful for grouping source files together so that common build options can be set for the whole group. They generally contain one or more source files. Contained files are compiled and linked with the program or library which includes the subproject. Subprojects are also used to include the same group of files in two different projects.
Select one	Sometimes a program needs to work on different kinds of hardware platforms. Usually, most of the source files are identical, but there may be one or more assembly language routines that vary from processor to processor. **Select one** files allow the same **.bld** file to build the program by selecting one of several assembly language files. For information about setting up **Select one** files, see "Building platform-specific programs from the same source files" on page 25.
Single file library	A single object file used as a library.

Type (drop-down list)	
File Type	**Meaning**
Library	A normal library, usually a build file (*.bld). A build file (*.bld) of this type contains source files and/or subprojects. When the library build file is built, the source files are compiled and linked to create the library. If the library build file is included in a program, then building the program also builds the library and links the library into the program. See “To link to a library that gets built with your project” on page 16 for more information. On the other hand, a library file (*.a, *.lib, *.dll, etc.) may also have this type. A library file with this type must be included in a program. Building the program will link the library file into the program. See “To link in a compiled library” on page 16 for more information.
Shared library	A shared library, usually a build file (*.bld). When building shared libraries, it is necessary to build a corresponding shared data library containing the modules of the shared library that exports initialized data. MULTI creates both types of libraries from the same build file. To create a build file with all of the desired library files, mark each shared data file by selecting the **Includes shared data** option in the File Options dialog box, General tab. If you set the file type to **Shared library** in the File Options dialog box, General tab, then files are built whether or not they are marked as including shared data. If the file type is set to **Shared data library,** then only the files marked as including shared data are built.
Shared data library	A shared data library, usually a build file. Refer to **Shared library** above for more information.
Include file	An include file. Adding include files to the project does not affect the actual include files included during a compile. See “To define header files” on page 17 for more information.
Script	Contains shell script commands.
Documentation	A documentation file.
C source file	Contains C source code. During a build, the file type is ignored and the Builder uses the file’s extension to determine the correct compiler to use. For example, setting the file type to C Source File for a file named foo.f still runs the Fortran compiler.
Fortran source file	Contains FORTRAN source code.
Pascal source file	Contains Pascal source code.
C++ source file	Contains C++ source code.
Ada source file	Contains Ada source code.
Object file	An object code file. The object file will be linked into the program or library containing it.
Linker file	A linker directives file. This file is passed to the linker.
Assembly file	Contains assembly language code. The assembly file will be assembled and linked into the program or library containing it.
Integrate file	Integrate configuration file used for the INTEGRITY RTOS.

Type (drop-down list)	
File Type	**Meaning**
Custom	A source file which needs custom build rules. To set the custom build rules from the Builder, choose Project > Options for *Selected Files* and choose the Configuration tab. Scroll the commands list down to Custom Processor, and set the appropriate command to build your custom source files. See “File Options > Configuration tab” on page 57 for more information.

Show (drop-down list)

The Show drop-down list box sets which levels of information are displayed when building, but the Build Panel overrides this which precedes the selected level in the list. Any of these settings automatically set all the choices before it in the list. For example, setting Warnings automatically sets Progress and Errors

Default

Uses the inherited setting. The default is **Warnings**.

Errors

Shows errors from the Builder, compiler, and linker.

Progress

Shows the steps being executed. For example, files being compiled or your program being linked.

Warnings

Shows warnings from the Builder, compiler, and linker. The MULTI default is to build without warnings.

Dependencies

Shows why actions occur. For example, why a particular source file is being recompiled (because a dependent source file has changed.)

Commands

Displays the compiler driver command lines used by the builder to run the compiler, assembler, and linker as they are executed without actualling run the tools. Equivalent to the **-v** and **-#** build-time options. For example:

Debug

Shows warnings from the Debugger.

Debugging level (drop-down list)

The debugging level drop-down list box sets the type of debugging information the compiler outputs.

Default

Uses the inherited settings. The default is **MULTI**.

None

No debugging information should be generated.

Stack

Generates a stack frame in every routine to support stack traces. Without this option, some routines may not create stack frames, thereby reducing code size and improving performance. This option does not imply the **-g** option, but is implied by the **-g** option. This option also disables leaf procedure optimization. Equivalent to the **-ga** debugging option.

Plain

Generates source-level symbolic debugging information. The debugging information varies with the capabilities of the target system. Debugging languages other than C has severe limitations. Equivalent to the **-g** debugging option.

MULTI

Generates extended debugging information for MULTI. This option is required for proper debugging of Ada, C, C++, FORTRAN, and Pascal code, since it provides information for handling language specific features. Equivalent to the **-G** debugging option.

Performance analysis (drop-down list)

Sets the level of profiling information output by the compiler.

Default

Uses the inherited settings. The default is **None.**

None

No profiling information is generated.

Percent

Generates profiling calls. When your source files are compiled with this option, calls to routines for maintaining profiling information are embedded in each routine in the source file. This displays the percentage of time spent at each source line. Equivalent to the **-p** build-time option.

Functions

Determines the number of calls to each function. This option also sets **Percent**. Equivalent to the **-p** build-time option.

Graph

Generates code to collect extended profiling information to include call graph information. This option functions similarly to the **-p** option,

except additional information collected produces a call graph report. This option also sets **Functions** and **Percent**. Equivalent to the **-pg** build-time option.

Coverage analysis (drop-down list)

Coverage analysis generates basic block profiling calls. When your source files are compiled with this option, calls to routines for maintaining profiling information are embedded in each block of instructions generated by the compiler. This displays information about which source lines are actually used when your program executes. This is either turned **On** or **Off**. The default is **Off**. Equivalent to the **-a** build-time option.

Automatically use MVC

Automatically places all text files under MULTI Version Control (MVC) when added to the project list or edited. If a non-existent file is added to the project list, it is not placed under version control until edited and created.

Driver options:

Any extra options sent to the driver are listed here. The *Development Guide* for your system provides a description of driver options.

Defines:

Enters any desired macro definitions for the preprocessor. However, do not put a **-D** in front of your entries. Equivalent to the **-D** C and C++ option. For example:

xxx=1,yyy=444

this defines **xxx** as 1 and **yyy** as 444.

Undefines:

Does not convert uppercase user-supplied variables to lowercase. By default, FORTRAN is not case sensitive and all FORTRAN names convert to lowercase. The compiler and library both assume this translation is performed. This option generally accesses variables defined in C as uppercase. However when using this option, all FORTRAN keywords must be lowercase, making the compiler incompatible with the ANSI FORTRAN-77 standard. Equivalent to the **-U** FORTRAN option. Placing a minus sign (-) prevents any symbols, such as **__STDC__** and **__ghs** to be defined by the Builder or compiler. Note that there are two underscores (_) before and after '**STDC**', and two underscores before '**ghs**'.

Libraries:

Specifies extra user libraries. Library names are either given with their full pathname, or a simple name, such as **foo**.

If you specify the file extension or full path of the library, the Builder looks for the library in the local directory rather than the directories specified by the Library Directories field.

If you enter a simple name, such as **bat**, the Builder looks for **bat.a** in the library directory paths set in the Library Directories field.

To indicate the library located in the local directory, use **./***library*, for example, **./***bat*.

This box only adds libraries. To replace system libraries or Green Hills libraries such as **libansi**, edit the Advanced tab of the File Options dialog box.

Source directories:

Normally, the Builder looks for source files in the same directory as the build file and in the directory from which the Builder is run. Additional directories specified here are searched for source files as well as include files. This is an editable field in the File Options dialog box, General tab.

Library directories:

Additional library search paths may be listed here.

Don't rebuild because of changes in:

This file

By default, all files depend on the Builder file which includes them. For example, if a program **fly.bld** contains a source file **bat.c**, then **bat.c** depends on **fly.bld**. So, if **fly.bld** changes, then **bat.c** along with every other file is rebuilt. This is not always desired. When this option is set, the Builder does not consider changes in the selected file and does not rebuild other files when it changes.

Include files

By default, the Builder determines dependencies of source files on the files they include (such as header files). Therefore, changing an include file causes the recompilation of all the source files that include it.

When this option is set, the Builder ignores all dependencies on include files for the current file.

Other files:

This field allows you to enter files which you do not want automatically rebuilt because changes have been made to them. For example, if you want the current file to depend on every header file except **fly.h**, then type **fly.h** in this box.

File Options > Optimization tab

This tab controls the compilers' optimization features. These optimizations are discussed in more detail in the Optimization chapter in the *Language User's Guides*.

Default

Uses the inherited settings. If no parent project has an explicit setting, then the default is **No optimization**.

No optimization

No optimizations should be performed. Any settings in the Advanced Optimizations dialog will be ignored.

Optimize for size

Employ optimization strategies that reduce code size, potentially at the expense of code speed.

Optimize for speed

Employ optimization strategies that favor code speed, potentially at the expense of code size.

Advanced button

If either "Optimize for size" or "Optimize for speed" are selected, the Advanced button will be available. Pressing the button will open the Advanced Optimizations Options dialog box.

Advanced Optimizations Options dialog box

To get to this dialog box, do this:

1. From the Builder, choose Project > Options for *Selected Files*....
2. Choose the Optimization tab.
3. Choose either "Optimize for size" or "Optimize for speed".
4. Click Advanced...

Many of the options in this window automatically set other options in this window. Use Merge to display the fully resolved option settings. The following table shows the name of the section in the *Language User's Guides* dealing with the given optimization:

Optimization	**Section**
Inline	Inlining Enabled with -OI
Loop	Loop Optimizations Enabled with -OL

The following table shows the minor optimizations and the corresponding command line option. Refer to the description of optimization control in the processor specific *Development Guide* for more information.

Minor Optimization	**Command Line Option**
Peephole	-Onopeep
Common subexpression elimination	-Onocse
Constant propagation	-Onoconstprop
Unroll loops	-Onounroll
Recognize min, max, abs expressions	-Onominmax
Pipeline scheduling	-Onopipeline
Inline strcpy() and strcmp() calls	-Onostrcpy
Tail recursion	-Onotailrecursion
Unroll loops up to 8 times	-Ounroll8
Unroll bigger loops	-Ounrollbig
Pipeline only within source line	-Olimit=pipeline
Peephole only within source line	-Olimit=peephole

The optimizations in the following table have an associated command line option in the “register allocation by coloring” description in the *Language User’s Guide*:

Minor Optimization	Command Line Option
Allocate auto variables in registers	-autoregister
Overload variables in registers	-overload

The first table of optimizations contains all major categories, while the second and third tables contain minor optimizations. Turning on some of the major optimizations automatically turns on some of the minor optimizations. The minor optimization boxes are not altered to reflect their new state. To see the final state of the minor optimizations, use the **Merge** button in the Builder window. The following table explains the state of minor optimizations:

Minor Optimization	State
Overload registers Auto register	Always on, unless forced off.
Loop unrolling	Implied by **Loop** optimization.
Unroll 8 Unroll Big	Requires **Loop** optimization, but always off unless forced on.
All others	Always off, unless forced on.

Most of the time, you will be turning off optimizations, since **Optimize for size** and **Optimize for speed** turn on most optimizations by default. The minor optimizations are useful in special circumstances.

The textfields allow you to enter additional information about the corresponding optimizations. For example, you may enter functions to inline in the **Inline** textfield, and functions to loop-optimize in the **Loop** textfield.

There are three ways to use the **Inline** field.

- Click the small box to the left. The compiler inlines only those functions it determines heuristically to be good inlining candidates.
- Enter a list of functions without clicking the box. The compiler inlines only those listed functions.
- Enter both a list of functions and click the box. The compiler inlines the listed functions and determines what other functions to inline.

The **Loop** option works differently. If you click the checkbox, then all functions will have loop optimizations applied, overriding any functions listed in the textfield. So, if you enter any specific functions in the textfield, do not click the checkbox to the left.

Note: Not all optimizations are supported for all targets.

File Options > Run-time Error tab

(Builder: Project > Options for *Selected Files*... > Run-time Error tab)

Select the options that enable the desired error checks. Most of these checks occur at run-time, although some occur at compile time.

Memory checking (drop-down list)

Default

Maintains the previous or inherited setting. Originally, the default is **None**.

None

Disengages memory checking.

Allocation

Equivalent to the **-check=alloc** command line option. Enables the Debugger's **findleaks** command and checks for the following memory errors. (See also "Finding memory leaks" in Debugging with MULTI 2000.)

If the program attempts to free memory not previously allocated, this error is reported:

```
Attempt to free something not allocated
```

If the program attempts to free memory already free, sometimes the previous error message is reported here. Otherwise, this error is reported:

```
Attempt to free something already free
```

If the program attempts to allocate memory after various other errors occurred, this error report appears:

```
Malloc internals (free list?) corrupted
```

Memory

Generates an error message when the program tries to access memory that is not yet allocated. (Equivalent to the **-check=memory** command line option.) It displays the appropriate **Allocation** error messages, above, in addition to the following:

```
Attempt to read/write memory not yet allocated
```

Array Bounds

Checks array indexes against array bounds. For constant indexes, this check occurs at compile-time; for other expressions at run-time. (Equivalent to the **-check=bounds** command line option.)

The error message is:

```
Array index out of bounds
```

Assignment Bounds

When assigning a value to a variable or field which is a small integral type such as a bit field, this checks if the value is within the range of the type. (Equivalent to the **-check=assignbound** command line option.) The error messages are:

```
Assignment out of bounds

or
Value outside of type
```

NULL Dereference

Generates an error message for all dereferences of NULL pointers. (Equivalent to the **-check=nilderef** command line option.) The error message is:

```
NULL pointer dereference
```

Case/Switch Statement

Generates a warning if the **case/switch** expression does not match any of the **case/switch** labels. This does not apply when using a default **case/switch** label. (Equivalent to the **-check=switch** command line option.) The error message is:

```
Case/switch index out of bounds
```

Divide by Zero

Generates an error message indicating a divide by zero. (Equivalent to the **-check=zerodivide** command line option.) The error message is:

```
Divide by 0
```

Unused Variables

Generates an error message at compile-time for declared variables never used. (Equivalent to the **-check=usevariable** command line option.) The error message is:

```
Unused variable
```

Pascal Variants

Checks that the tag field of a variable declared as a variant record type matches one of the case selectors in the record. This applies only to Pascal. (Equivalent to the **-check=variant** command line option.) The error message is:

```
Bad variant for reference
```

Watchpoint

Enables the Debugger command **watchpoint** to create one watchpoint without using an assertion. (Equivalent to the **-check=watchpoint** command line option.) See also "watchpoint" in Debugging with MULTI 2000. The error message is:

```
Write to watchpoint
```

Return

Generates a warning if a non-void procedure ends without an explicit return. For example, the following procedure generates a warning when exiting:

```
int func() {
    for (int x = 0; x< 10; x++)
    {
        if (x == 10)
        return x;
        }
}
```

This option only applies to C and C++. (Equivalent to the **-check=return** command line option.) The error message is:

```
No value returned from function
```

File Options > Configuration tab

(Builder: Project > Options for *Selected Files*... > Configuration tab)

Builder:

Specifies the name of the Builder program (**build**). This program is otherwise assumed to be in the same directory as MULTI.

Select:

For the project type Select One, you may list several files which have different extensions. Only one of those files will be built, however. To determine which one is built, the builder looks at the ‘select’ list. This is a list of suffixes which are to be built. Thus, when the builder comes across a Select One subproject, it will look through the ‘select’ suffix list, and find the file which has a valid suffix.

Green Hills C++ include dirs:

A list of directories containing Green Hills C++ include files. Adds *directory* to the list of directories to search when processing **#include** directives. If **file.cxx** contains the line **#include “header.h”**, the compiler searches for **header.h** in the **file.cxx** directory, then in directories specified in Source Directories on the File Options dialog box, General tab, and finally in a list of default directories. If **file.cxx** contains the line **#include <header.h>**, the processing is the same except the directory containing **file.cxx** is not searched. If **header.h** is specified with an absolute pathname, then no directories are searched. Equivalent to the **-I** and **-YI** build-time options.

Green Hills C include dirs:

A list of directories containing Green Hills C include files. Adds *directory* to the list of directories to search when processing **#include** directives. If **file.c** contains the line #include **“header.h”**, the compiler searches for **header.h** in the **file.c** directory, then in directories specified in Source Directories on the File Options dialog box, General tab, and finally in a list of default directories. If **file.c** contains the line **#include <header.h>**, the processing is the same except the directory containing **file.c** is not searched. If **header.h** is specified with an absolute pathname, then no directories are searched. Equivalent to the **-I** and **-YI** build-time options.

System include dirs:

A list of directories containing system include files. Equivalent to the **-YI** build-time option.

Green Hills library dirs:

A list of directories containing Green Hills libraries. Equivalent to the **-L**, **-YL**, and **-YU** build-time options.

System library dirs:

A list of directories containing system libraries. Equivalent to the **-L**, **-YL**, and **-YU** build-time options.

Tools directory:

Changes the default directory to look for the commands in the list described below under **Commands**.

Alternate tools dir:

If you select the Gnu tool chain or a UNIX tool chain, instead of the Green Hills tool chain, then this directory is used for the assembler, linker, and other files that comprise this alternate tool set.

Commands:

Displays many different commands called by the Builder. You can change the name and arguments to any or all of these commands. Select the command and then fill in the appropriate information in the text fields below the list.

Generally, the Builder's default information is correct so you can leave these entries blank.

Command directory:

Specifies a new directory to search for a command.

Command name:

Specifies a new name for a command.

Arguments:

Specifies a list of additional arguments sent to the command when called.

File Options > Actions tab

(Builder: Project > Options for *Selected Files*... > Actions tab)

Output Filename:

Names the output file of the current driver command. In the simplest case, the linker output file is called *filename*. If another file is generated from the linker output file, such as an S-Record file, this determines the name of the last file created, which is the S-Record file. If only one source file is named, **-o** is used with either the **-S** or **-c** option to name the output of the compiler or assembler. Equivalent to the **-o** build-time option.

On a project of type **Program**, **Library**, **Shared library**, or **Shared data library**, these commands are executed before calling the linker or archiver, and not at the very beginning of processing. However, projects of type **Shared** and **Nobuild** generate no output files, and at this time this field is ignored for projects of these types.

Append Extension:

To make your file portable across different systems, the Builder can append the appropriate extension to the output filename, so you do not have to specify it yourself.

For example, if you are building a program with a build file called **fly.bld** and want to name the executable file **bat**, enter **bat** into the output filename box. Once this is done, the executable will call **bat** on every system, including those expecting the executables to have extensions such as .**bat.exe**. When setting this option, the Builder adds the appropriate extension, .**exe**, on those systems that require it.

Object Directory:

Equivalent to **-object_dir**.

Stop with (drop-down list)

The **Stop with** field should be used in conjunction with the **C Source** option in the **Compilation** menu, located in the Advanced tab. **Stop With** sets the type of file with which to stop when building, so that you can look at the intermediate files. For example, if you have a C source file and want to compile it into assembly code but you do not want it to process any further, set this menu to **Assembly**. The Builder compiles source files to object files by default.

Some of the file types in this menu are incompatible with the type set in the **Type:** menu in the File Options dialog box, General tab. For example, you cannot make a C source file into an archive file.

Inf file

Indicates that an analyzing pass is being performed, and stops after generating the **.inf** (information) file. The **.inf** file contains reader-file dependent information.

Preprocessor output

For source files which are preprocessed (C and C++ files plus preprocessed assembly language files), it runs the compiler only as a preprocessor and places the output in the standard output file. Although this does not process files as quickly as a standalone C

preprocessor, it duplicates the action that is taken when the file is compiled. Equivalent to the **-E** C and C++ preprocessor option.

Preprocessor file

For source files which are preprocessed (C and C++ files plus preprocessed assembly language files), it runs the compiler as a preprocessor but sends the output to a new file with a **.i** extension. Equivalent to the **-P** C and C++ preprocessor option.

Translated C file

When converting from other languages to C, this stops with the C file. This option is used in conjunction with the C Source option from the Compilation menu. This is currently not supported.

If you are converting from C++, you can set the Leave Translated C in the C++ option window to achieve the same effect.

Syntax

For Ada, C, C++, FORTRAN, and Pascal, it checks the syntax of the source file, but does not generate code. Equivalent to the **-syntax** command line option.

Assembly

Only produces an assembly file from the source file. For each source language file specified in Ada, C, C++, FORTRAN, or Pascal, compiles the file into an assembly language output file using standard naming conventions. Implies the **Assembly** option in the **Compilation** menu. Equivalent to the **-S** command line option.

Object

Only generates a relocatable object file for each source input file with a filename of *inputfile***.o**. Applies to all source files which are compiled, in Ada, C, C++, FORTRAN, and Pascal. Equivalent to the **-c** command line option.

Executable

Stops with an executable file. This option applies to projects of type **Program**. If the Debugging level is not MULTI, this option has no effect. Select this option to prevent **dblink** from running on the executable after it is generated by the linker.

Sym file

Runs **dblink** on the executable file, producing debugging symbol files and strips the original executable file of all symbol information. This option applies to projects of type **Program**. It is the default with Debugging level: MULTI.

Archive

Runs the librarian to generate an archive instead of invoking the linker to generate an executable program. It is the default for projects of type **Library**. This option must be used with the **-o** *filename* option with a **.a** extension. For example:

```
cc960fly.c -archive -o libfly.a
```

Equivalent to the -**archive** command line option.

Shared Object

Stops with a shared library file with a **.so** extension. This option applies to project of type **Shared Library** and is the default for such projects. This type of project is only supported if the Target OS is UNIX. Equivalent to the -**shared** command line option.

Shared Data

Stops with a shared data library file with a **.sa** extension. This option applies to projects of type **Shared Data Library** and is the default for such projects. This type of project is only supported if the Target OS is SunOS.

Dependencies:

Enters any additional files you want the current file to depend on. When changes are made in any of these files, the Builder rebuilds the current file.

Commands to set up input files:

If you want to execute shell commands before compilation, enter a list of executable commands. Typically, you run commands to set up source files such as preprocessing. These commands are executed verbatim with the default command processor with no variable substitution.

Commands to process output:

If you want to execute shell commands after compilation, enter a list of such commands here. Typically, you run commands to process the output of the compilation and produce more suitable output. These commands are executed verbatim with the default command processor with no variable substitution.

File Options > Advanced tab

(Builder: Project > Options for *Selected Files...* > Advanced tab)

Set by the target Build File, the options on this tab are usually already calculated for you by the Builder. Changing these options will have unspecified results as they are not valid in all configurations. All of these options are only set in projects of type **program** or **nobuild**. This ensures consistency across all files in a single program. In addition, many of these options apply to the link phase and are ignored if set on individual files within a program.

Processor (drop-down list)

Sets the processor family for the program being built. This selection affects the behavior of the Builder in many ways. Not all options are relevant for all processor families. In particular a sub-window in the CPU Options dialog is provided for each processor family. Only the options in the sub-window corresponding to the selected processor take effect.

When set to **C Translator**, the source code is translated into C by one of the C translators. The native C compiler is then run on the generated C code.

Compilation (drop-down list)

Compilation specifies the output format of the compiler itself.

C Source

The source code is translated into C by one of the C translators. The native C compiler is then run on the generated C code.

Assembly

The source file produces an assembly language file. The assembler will then run on that file to produce an object code. Equivalent to the **-noobj** command line option.

Object

The source file directly produces an object code file without producing an assembly language file. Currently, this is only supported on M16, 68000 and SH processors. Equivalent to the **-obj** command line option.

Alignment (drop-down list)

Alignment sets the maximum data alignment for the target. This is rarely used. Equivalent to the **-align=** machine specific option.

Structure packing (drop-down list)

Specifies the maximum alignment of fields in a structure. This feature is NOT supported for all processors.

Toolchain (drop-down list)

This feature sets the toolchain: the assembler, linker, libraries and utilities. The Builder and compilers are adapted to work with assemblers and linkers from various vendors. This field should not be set or changed at any time.

Object format (drop-down list)

BSD

Used by SunOS and some Gnu compilers. Equivalent to the **-bsd** command line option.

COFF

The Common Object File Format is used by UNIX System V.3 and older embedded environments. Equivalent to the **-coff** command line option.

ELF

The Executable and Linking Format is used by UNIX System V.4 and the modern embedded environments. Equivalent to the **-elf** command line option.

Oasys 68k

A proprietary format generated by the Oasys 680x0 assembler and linker. Equivalent to the **-oasys** command line option.

Output mode (drop-down list)

This sets the file format produced by the linker. Many environments only support one binary format. The following are descriptions of these output formats:

BSD

Used by SunOS and some Gnu compilers. Equivalent to the **-bsd** command line option.

COFF

The Common Object File Format is used by UNIX System V.3 and older embedded environments. Equivalent to the **-coff** command line option.

ELF
The Executable and Linking Format is used by UNIX System V.4 and the modern embedded environments. Equivalent to the **-elf** command line option.

S-Records
Produces a COFF or ELF file, translates into an S-Record file, and keeps both files. Equivalent to the **-srec** linker option.

Oasys 68k
A proprietary format generated by the Oasys 680x0 assembler and linker. Equivalent to the **-oasys** command line option.

Memory
Memory image format. Equivalent to the **-memory** command line option.

HP/OMF
Obsolete.

Only S-Records
Produces a COFF or ELF file, translates into an S-Record file, and then deletes the COFF or ELF file. Equivalent to the **-sreconly** linker option.

Oasys Objects and S-Records
Produces S-Record output directly using Oasys object format instead of COFF or ELF. Equivalent to the **-srecoasys** command line option.

Tek Hexadecimal
Tektronix extended hex format (680x0 only).

Exormacs
Obsolete.

IEEE-695
Portable format used by 680x0 emulators. Equivalent to the **-ieee695** command line option.

Target OS (drop-down list)

Target OS sets the type of operating system on the target system you are building.

Temp Directory

Stores temporary files in the directory specified by *dir* instead of **/tmp**. This is useful if **/tmp** is on a small file system that may run out of disk space during

compiles with inlining or template processing. This is also set with the **TMPDIR** environment variable. For example:

```
setenv TMPDIR /usr/tmp
```

Equivalent to the **-tmp=** C compiler option.

Start address:

Specifies where the program starts. This is passed to the linker and is normally a symbol name, so it should be written so that the linker recognizes it (that is, you may need to include an extra underscore (_). This is equivalent to the **-entry=** linker option.

Start/End file dir

Contains startup files, such as **crt0.o**. Equivalent to the **-YS** command line option.

Start files

A list of files linked at the beginning of your program. For example: **mcrt0.o, crt10.o**. The files listed replace **crt0.o**. If you just want to suppress the linker from including the default startup code from the library into your program, you can type a '-' in this field. It will prevent the builder from linking in any startup code.

End files

A list of files linked at the end of your program. On systems using it, **crtn.o** is replaced by these files.

Green Hills libraries

Replaces the default Green Hills Libraries normally linked in with those listed here. These libraries are normally chosen automatically.

System libraries

Replaces the default System Libraries normally linked in with those listed here. These libraries are normally chosen automatically.

Remote

This field is only used in a **default.bld** file or a program build file. When the Builder window is opened on a build file with this field set, the value specified

automatically loads into the text field next to the **Remote** button in the Builder window.

Small printf without %e%f%g

Uses libnoflt.a, a smaller version of **printf,** that does not handle floating point numbers. The I/O routines within this library do not contain instructions for floating point support and therefore are much smaller.

Show headers

Displays a list of files opened by a **#include** directive. Equivalent to the **-H** command line option.

Source lines in asm File

Outputs lines from original source files into the assembly language output of the compiler as comments. This option has no effect with direct binary code generation. This option interferes with some optimizations, including loop optimization which produces inferior code in some circumstances. Equivalent to the **-passsource** command line option.

Show Versions

Displays the copyright banner and version number of the compiler, assembler, and linker as they are run. By default, the version and copyright banner is suppressed. Equivalent to the **-V** command line option.

Put versions

Places the compiler's version into the comment section of each object module. Currently, this is only available on some UNIX systems. Equivalent to the **-Qy** command line option.

Output dual debug formats

Output both Green Hills proprietary information as well as Dwarf.

Dynamic download project

For the INTEGRITY RTOS, build project for dynamic download.

Keep temp files

Does not delete temporary files after they are used.

Link without default startfiles or libraries

Prevents the Builder from adding any of the following to the link command:

- Green Hills Libraries (Advanced tab)
- System Libraries (Advanced tab)
- Green Hills Library directories (Configuration tab)
- System Library directories (Configuration tab)
- Startup Files (Advanced tab)
- End Files (Advanced tab)

The option still specifies libraries and library directories using the fields in the File Options dialog box, General tab. A linker directive file still adds to the link command line with the Oasys 68000 linker, **l68**. However, it does not add any startup files or any libraries and passes fewer options to the linker command line. (This is equivalent to the **-nostdlib** command line option.) An alternative is to override the default for individual fields listed above by placing a dash "**-**" in the field.

Languages Used:

These check boxes tell the Builder which languages are used that it does not recognize, such as those compiled to object files that have no source. The Builder uses this field to select libraries during the link phase. This is equivalent to the **-language=** linker option which ensures that the driver is aware of all languages in use. It is specified once for each language used other than C.

Language Options dialog box

(Builder: Project > Language Options for *Selected Files*...)

The Language Options dialog box contains several tabs, each of which contains options for a particular language. Most of the options are compile-time options described in the Green Hills *Language User's Guides* and the *Development Guides*.

For an overview of issues involved when setting options, see "Setting options: An overview" on page 20.

Language Options > C tab

To get to this tab, from the Builder, choose Project > Language Options for *Selected Files*..., and choose the C tab.

The following are descriptions of the items in the C tab.

C version (drop-down list)

K+R

For C source files, interpret the source code as the C version documented in Kernighan & Ritchie, first edition, and compatible with the portable C compiler, or PCC. Equivalent to the **-k+r** and **-Xs** C options.

Transition Mode

Selects a mode of ANSI C compatibility similar to AT&T C Issues 5.0 transition mode supporting function prototypes and the new ANSI keywords **signed** and **volatile** in a non-ANSI environment. This is the default. Equivalent to the **-Xt** C option.

ANSI

Sets the compiler in Permissive ANSI compatibility mode. With some systems, this uses the header files in **/usr/green/include** and **/usr/green/ansi** before those in **/usr/include**.

This mode supports the language features of the ANSI X3.159-1989 standard, while allowing certain useful but non-compliant constructs in an ANSI C framework. Equivalent to the **-ansi** C option.

Strict ANSI

Strict ANSI mode is 100% compliant with the ANSI X3.159-1989 standard and does not allow non-standard constructs. This also uses the header files in **/usr/green/include** and **/usr/green/ansi** before those in **/usr/include**. Equivalent to the -**ANSI** C option.

Type of wchar_t (drop-down list)

Specifies the type of **wchar_t**, the type of all wide-characters. This allows you to set the size of all wide characters and determine whether they are signed or unsigned. Equivalent to the -**shortwchar** and **-signedwchar** C and C++ compiler options. The type selected here has the following effects:

- Changes the size and signed or unsigned attributes of wide character constants and strings such as **L'x'** and **L"Hello"**.
- Predefines one of the following symbols:
  __WChar_Is_Unsigned__
  __WChar_Is_Signed__
  and also predefines one the following symbols:

__WChar_Is_Short__
__WChar_Is_Int__
__WChar_Is_Long__
__WChar_Is_LongLong__

Note that there are two underscores (_) both at the beginning and at the end of each of the above symbols.

- Selects which type is used for the *typedef* of **wchar_t** in **stddef.h**.

Only the default selection works with the libraries provided.

short
Default for SunOS 4.x.

unsigned short
int
Default for Ultrix on DEC station.

unsigned int
long
Default on UNIX System V.4, Solaris, and all embedded products.

unsigned long
Not supported for all target processors. See your target *Development Guide* for more information.

long long
Not supported for all target processors. See your target *Development Guide* for more information.

unsigned long long
Not supported for all target processors. See your target *Development Guide* for more information.

Target kanji (drop-down list)

Host kanji (drop-down list)

The Target kanji and Host kanji drop-down list boxes control the internal representation of kanji characters recognizable in C host code, comments, and character strings. If Host kanji > EUC and Target kanji > Shift-JIS are both selected, then the compiler automatically translates character string literals from EUC to Shift-JIS format. The combination of Host kanji > Shift JIS and Target kanji > EUC is not supported. On SunOS, Host kanji defaults to EUC. On Windows and HP/UX, Host kanji defaults to Shift-JIS. Target kanji defaults to Host kanji. Equivalent to the **-kanji=** command line option.

The following are descriptions of the check boxes in the **C** tab.

Ignore Duplicate #include

Ignores an **#include** directive if attempting to include a file already included. The file must appear with exactly the same name in both **#include** directives to ignore the second **#include** directive.

If a filename appears in more than one **#include** directive during a single compilation, it skips all of the directives except the first one. Equivalent to the **-includeonce** C preprocessor option.

Ignore All #include

Ignores all **#include** file directives. Equivalent to the **-includenever** C preprocessor option.

Allow Macros to be Re#defined

Suppresses the warning or error normally given when two **#define** directives provide different values for the same preprocessor symbol. Equivalent to the **-redefine** C preprocessor option.

Allow Wrong #directives inside #if 0

During preprocessing, lines inside of false **#if**, **#elif**, **#ifdef**, and **#ifndef** are ignored. With the exception that a warning or error is given for lines beginning with **#,** they do not contain legal preprocessor directives. This option suppresses these warnings and errors. Equivalent to the **-nocpperror** C and C++ preprocessor option.

Warn for Unknown #pragma

Generates a warning for unknown **#pragma** lines. Normally, unknown **#pragma**e are ignored silently. Equivalent to the **-unknownpragmawarn** command line option.

No Warning for Incorrect #pragma

Suppresses warnings for errors in **#pragma** that are recognized by the compiler and that are incorrect. Equivalent to the **-nopragmawarn** C and C++ preprocessor option.

Allow #pragma asm and #pragma inline

Allows the use of **#pragma asm**, **#pragma endasm**, and **#pragma inline** in C source files. See the *Green Hills C User's Guide* for more information on these pragmae. Equivalent to the **-pragma_asm_inline** command line option.

No Output for #ident or #pragma ident

Prevents the compiler from outputting an **ident** directive in the assembly language output or from placing the same information in the **.comment** section when generating COFF or ELF object files directly. This option is primarily intended for an assembler or linker that does not support the **ident** directive. Equivalent to the **-noidentoutput** C compiler option.

Allow // style comments in C

Allows C++ style comments (beginning with // and terminating with a new line to be used in C). This option is also used with preprocessed assembly files. Equivalent to the **-slashcomment** command line option.

Keep Comments in Preprocessor Output

Includes comments in the preprocessor output. The default strips comments from the output. Equivalent to the **-C** C and C++ preprocessor option.

Concat 2 Symbols Separated by Comment

Allows /* */ as concatenation in K&R C. You can turn off this option with the **-Zconcatcomments** option. Equivalent to the **-concatcomments** C option.

Warn for Function Used without Prototype

Generates a warning if a function is referenced or called but no prototype is given for that function. This is an extension to ANSI C, ensuring that prototypes exist for all functions used. Equivalent to the **-wantprototype** C compiler option.

Disallow Function Used without Prototype

Generates a fatal error if a function is referenced or called but no prototype is given for that function. This is an extension to ANSI C, ensuring that prototypes exist for all functions used. Equivalent to the **-needprototype** C compiler option.

Allow ‘noalias’ keyword in C

Adds **noalias** keyword to C. You can turn off this option with the **-Znoalias** option. Equivalent to the **-noalias** C option.

Disable ANSI aliasing rules

Disables ANSI aliasing rules. Equivalent to the **-no_ansi_alias** build-time option.

No Warning for asm()

Does not give warnings for **asm** statements. Equivalent to the **-asmwarn** build-time option.

Do not reserve asm keyword

By default, the compiler recognizes **asm** as a keyword and gives a syntax error if any variable, structure field, macro, or function has the name **asm**. Selecting this option causes the compiler to treat **asm** as an ordinary identifier in C. Any attempt to use an **asm** statement with this option enabled causes the compiler to call a function **asm()** with a character string as its argument.

This option is implied by Strict ANSI mode. The **_ _asm** keyword is always recognized; only the **asm** directive without leading underscores is affected by this switch. This switch is enabled with **-ANSI**. Equivalent to the **-noasm** C compiler option.

Give fatal error for asm statement

Generates a fatal error if an **asm** statement is used. Equivalent to the **-asmillegal** command line option.

Allow Some Gnu Syntax Extensions

Supports GNU extensions, such as **#import**, zero size arrays, compound statements as part of expressions, inline functions, and the **_ _inline_ _** keyword. Equivalent to the **-gnu_c** C compiler option.

Japanese Automotive C

Enables a set of extensions to ANSI C used by Japanese automobile manufacturers. This option implies the following option settings:

```
Allow #pragma asm and #pragma inline
No warning for asm()
Do not reserve asm keyword
```

Refer to the *Green Hills C User's Guide* for more information on this option. Equivalent to the **-japanese_automotive_c** command line option.

Allow extern to be Initialized

Allows variables declared with the extern storage class to accept initial values. In the K+R definition of C this is an error, but is legal in ANSI C. This option only affects K+R. Equivalent to the **-initextern** command line option.

Disallow Old Fashioned Syntax

Does not recognize outdated syntax for initializing variables, such as **int i 5;**, and for assignment operators like **=+**, **=-**, **=*** . If this option is not set, these are accepted with a warning message. If this option is set, old fashioned initializations give the error:

```
expected '=' got constant
```

and an equal sign followed by the symbols:

```
+ - * / % & | ^ << >>
```

is recognized as two separate tokens. This results in a syntax error for the symbols:

```
+ / % | ^ << >>
```

but is correct for the symbols:

```
- * &
```

which are legal unary operators in C. Therefore, this option is required to correctly compile the following lines because no space appears after the equal sign:

```
int i, *p;
i =-3;
p =&i;
i =*p;
```

By default, this option is only set on native UNIX 68K compilers. In all other Green Hills compilers, this is turned off by default. Equivalent to the **-nooldfashioned** C option.

Use ANSI C Semantics for Assignment

Uses ANSI rules for **++** and ***=** in K&R C. Equivalent to the **-ansiopeq** command line option.

Allocate Small Enums as char or short

Allocates enumerated types to the smallest storage possible. Equivalent to the **-shortenum** C compiler option.

Consider char to be signed

Specifies type **char** as signed. Equivalent to the **-signedchar** C compiler option.

Consider Bit-fields to be Signed

Specifies a bit field whose type is **signed** to be interpreted as a signed quantity. Equivalent to the **-signedfield** C compiler option.

Consider Pointers to be Signed

Specifies pointers and addresses as signed. This is the default. Equivalent to the **-signedptr** C compiler option.

Truncate External Symbols to 8 characters

Truncates all symbol names to eight characters for compatibility with older compilers and linkers. Equivalent to the **-T** C compiler option.

Allocate unique space for all strings

Creates separate space for each string. Normally, the compiler performs an optimization in which equivalent strings are combined to share the same space. This reduces code size, but could cause problems if the strings are modified. This option disables the optimization and forces each string to have unique storage. Equivalent to the **-uniquestrings** C compiler option.

Language Options > C++ tab

(Builder: Project > Language Options for *Selected Files*... > C++ tab)

The following are descriptions of the drop-down list boxes in the C++ tab.

C++ version (drop-down list)

Standard C++

Enables ANSI C++ mode. Warning messages are issued when non-ANSI C++ features are used. Features that conflict with ANSI C or C++ are disabled. Equivalent to the **--std** command line option.

ARM

Accepts the C++ language as defined in *The Annotated C++ Reference Manual (ARM)*, by Ellis and Stroustrup. This version of C++ includes templates, exception handling, which must be explicitly requested, and

the anachronism of the book's Chapter 18. This is essentially the same language as the language reference manual for cfront 3.0, with the addition of exception handling. This is the default C++ version.

ESTL C++

Enables the extended embedded C++ dialect. Equivalent to the **--ee** command line option

Embedded C++

Enables the Embedded C++ dialect, with templates, STL, namespaces, and mutable, new-style casts. Equivalent to the **--e** command line option.

Cfront 3.0

Enables compatibility with cfront version 3.0. This causes language constructs to be accepted which are not necessarily part of the C++ language definition, but which are accepted by the AT&T C++ Language System (cfront) release 3.0. This mode also enables acceptance of anachronisms.

Cfront 2.1

Enables compatibility with cfront version 2.1. This causes language constructs to be accepted which are not necessarily part of the C++ language definition, but which are accepted by the AT&T C++ Language System (cfront) release 2.1. This mode also enables acceptance of anachronisms.

C++ Library (drop-down list)

C++ Library Option	Command
Standard C++ library with exceptions.	--stdle
Standard C++ library without exceptions.	--stdl
Extended Embedded C++ library with exceptions.	--eele
Extended Embedded C++ library without exceptions.	--eel
Embedded C++ library without exceptions.	--el
Embedded C++ library with exceptions.	--ele
cfront version 2.1 compatibility with compilation of C++. This causes the compiler to accept language constructs that, while not part of the C++ language definition, are accepted by the AT&T C++ Language System (cfront) release 2.1.	--cfront_2.1 -2.1
cfront version 3.0 compatibility with compilation of C++. This causes the compiler to accept language constructs that, while not part of the C++ language definition, are accepted by the AT&T C++ Language System (cfront) release 3.0. This option also enables acceptance of anachronisms.	--cfront_3.0 -3.0
Minimum Runtime Support Library	--minl

Inlining (drop-down list)

Selects whether function inlining should be done.

Max inlining
: Enables maximum inlining of function calls.

Max inlining unless debug
: Disables maximum inlining of function calls when debugging information is requested.

Inlining
: Enables minimal inlining of function calls.

Inlining unless debug
: Disables inlining of function calls when debugging information is also requested.

No inlining
: Disables inlining of function calls.

Virtual tables (drop-down list)

Controls the allocation of virtual tables.

Standard Allocation

Uses the standard heuristic to define a virtual function table for a class. The virtual function table for a class is defined in a compilation if that compilation contains a definition of the first non-inline pure virtual function for the class. For classes that contain no such function, the default behavior defines the virtual function table as a local static entity.

Force Allocation

Forces definition of virtual functions tables in cases where the heuristic used by the front end, to decide on the definition of virtual tables, provides no guidance.

Suppress Allocation

Suppresses definition of virtual function tables in cases where the heuristic used by the front end, to decide on the definition of virtual function tables, provides no guidance. For details on this heuristic, see **Standard Tables** above. The option suppresses definition of the local static virtual function tables.

Type of enum (drop-down list)

Selects the algorithm to allocate storage for enumeration types.

Int

Always allocates a full integer for an enumeration type. This is the default.

Smallest possible

Allocates enumerations to the smallest possible integral type.

Packing (drop-down list)

Selects the default alignment for packing classes and structs. This option is rarely used, and should match the alignment setting in the Project > Options for *Selected Files*... dialog box, Advanced tab.

The following items describe check box items in the Language Options dialog box, C++ tab:

Enable exception handling

Enables support for the C++ exception handling feature. Code size and speed may be impacted even when exception handling is not directly used.

Disable namespaces

Disables support for the C++ namespaces feature.

Enable std namespace

Enables implicit use of the standard namespace when standard header files are included.

Disable RTTI

Disables support for runtime type information (RTTI) features “dynamic_cast” and “typeid.”

Disable “bool” keyword

Disables recognition of the “bool” keyword. Equivalent to omitting the **--bool** command line option.

Disable “explicit” keyword

Disables support for the “explicit” specifier on constructor declarations. It is equivalent to the **--no_explicit** build-time option.

Disable wchar_t keyword

Does not recognize **wchar_t** as a keyword. Use this option if your source contains a **typedef** that declares **wchar_t.**

Disable array new/delete

Disables support for the array new and delete feature.

Recognize “restrict” keyword

Enables recognition of the “restrict” keyword.

Disable “extern inline”

Disable support for ’inline’ functions with external linkage in C++. Functions which are declared only ’inline’ will be external or static depending on the flag specified. Equivalent to the **--no_extern_inline** command line option.

Disable ’extern "C"’ type conversion

Disable an extension to permit implicit type conversion in C++ between a pointer to an ‘extern “C”’ function and a pointer to an ‘extern “C++”’ function. Equivalent to the **--no_implicit_extern_c_type_conversion** command line option.

C and C++ functions have distinct types

Function types are considered distinct if their only difference is that one has ‘extern “C”’ routine linkage and the other has ‘extern “C++”’ routine linkage. Equivalent to the **--c_and_cpp_functions_are_distinct** command line option.

Allow overloading of enum types

Allow operator functions to overload built-in operations on enum-typed operands.

Use late tiebreaker rules

When resolving an overloaded function, tie-breakers (‘const’ and ‘volatile’ qualifiers) are ignored during the initial comparison. They are considered only if the two functions are otherwise equally good on all arguments. Equivalent to the **--late_tiebreaker** command line option.

Force zero initialization of scalars

Force all uninitialized scalar global variables to be explicitly initialized to zero. Equivalent to the **--force_zero_initialization** command line option.

No constructor initialization in main

Do not generate an automatic call to _main from main. _main performs constructor initialization.

Enable multibyte characters

Enable processing for multibyte character sequences in comments, string literals, and character constants. Equivalent to **--multibyte_chars**.

Enable Microsoft extensions

Enables recognition of a set of Microsoft extensions. The *Green Hills C++ User’s Guide* discusses this in further detail.

Allow anachronisms

Enables support of anachronisms.

Use old for-loop initialization scoping

The old (Cfront-compatible) scoping rules means the declaration of a variable in the initialization part of a *for* statement is in the scope to which the *for* statement itself belongs. The new (standard conforming) rules, in effect, wrap the entire *for* statement in its own implicitly generated scope.

Don’t demangle linker messages

Does not demangle names that appear in linker messages. These are typically symbol names which are either undefined or multiply defined.

Leave translated C

Leaves a C version of the C++ code. The translated C file is *filename.ic*.

Keep comments in preprocessor output

If producing a preprocessor output file (see File Options dialog box, Actions tab) passes comment lines through from the C++ source to the preprocessor output file.

Ignore duplicate #include

Ignores an **#include** directive if attempting to include a file already included. The file must appear with exactly the same name in both **#include** directives to ignore the second **#include** directive.

If a filename appears in more than one **#include** directive during a single compilation, it skips all of the directives except the first one. Equivalent to the **-includeonce** C preprocessor option.

Consider char to be signed

The “char” type is treated as “signed char.”

Consider bit-fields to be signed

Bit fields within a struct or class are treated as signed entities by default.

Consider enum bit-fields to be signed

Bit fields of type “enum” within a struct or class are treated as signed entities by default.

Use long lifetimes for temps

Creates temporary variables whose lifetime ends at the earliest end of scope, end of switch clause, or next label. This is the behavior of cfront. Long lifetime temporaries are implied by the cfront compatibility modes. The alternative is for temporary variable lifetimes to end at the end of the full expression for which they are created. This is the behavior of standard C++.

Recognize alternate tokens

Enables recognition of alternative tokens. These are tokens that make it possible to write C++ without the use of the {, }, [,], #, &, |, ^ and ~ characters. The alternative tokens include the operator keywords, such as *and* and *bitand*, and digraphs.

More C++ Options > Template tab

(Builder: Project > Language Options for *Selected Files*... > C++ tab > More Options... > Template tab)

This dialog box contains more C++ specific options, including template, precompiled header, diagnostics, and listing file options.

Template mode (drop-down list)

Select the manner in which the compiler should force the template instantiation. This form of template instantiation is done when a source file is compiled, as opposed to allowing the automatic template instantiation mechanism to decide which templates need instantiation.

Never force instantiations
: Gives the automatic template instantiation mechanism complete control over which templates to instantiate.

Force instantiations for used entities
: The compiler instantiates all templates which are used in the current compilation.

Force all possible instantiations
: The compiler instantiates all template entities whether or not they have been used in this compilation.

Force local instantiations when used
: The compiler instantiates only the template entities that are used in this compilation, and forces those entities to be local to this compilation.

Disable automatic instantiations

Does not do template instantiation automatically. This assumes the responsibility of making sure the necessary entities are instantiated, possibly using one of the Template Mode settings, or through the use of #pragma directives.

Disable template implicit inclusion

"Implicit inclusion" is a convention that Cfront uses where template definitions must appear in a header (.h) file. For each such file, there must be a corresponding C++ source file containing the associated template definitions. This option tells the automatic instantiation mechanism NOT to attempt to use this convention to locate template definitions.

Use distinct template signatures

Uses signatures for template functions that can never match those of non-template functions. A normal (non-template) function, such as `void f(int)`, cannot be used to satisfy the need for an instantiation of a template, such as `void f(T)`, with `T` set to `int`.

Disable old-style specializations

Does NOT accept old-style template specialization, that is, specializations that do not use the **template** <> syntax.

Disable "typename" keyword

Disable recognition of the keyword 'typename'. 'typename' can be used instead of class when declaring template parameters. Equivalent to the **--no_typename** command line option.

Disable implicit typename determination

Disables implicit determination, from context, whether a template parameter-dependent name is a type or non-type.

Disable "guiding declarations"

Disable "guiding declarations" of template functions. A guiding declaration is a function declaration that matches an instance of a function template, but has no explicit definition (since its definition derives from the function template). For example:

```
template <class T> void f(T) { ... }
void f(int);
```

Equivalent to the --no_guiding_decls command line option.

Non-standard qualifier deduction

Controls whether non-standard template argument deduction should be performed in the qualifier portion of a qualified name.

One template instantiation per object file

Places each template instantiation in this compilation (function or static data member) in a separate object file.

More C++ Options > Precompiled Header tab

(Builder: Project > Language Options for *Selected Files*... > C++ tab > More Options... > Precompiled Header tab)

Automatic PCH processing

Automatically uses or creates a precompiled header file.

Disable PCH creation message

Disables the message indicating that a precompiled header file was created or used during a compilation.

PCH directory

Directory to search for and create a precompiled header file.

Create PCH file:

Creates a precompiled header file with the specified name.

Use PCH file:

Uses a precompiled header file of the specified name as part of the current compilation.

More C++ Options > Diagnostics tab

(Builder: Project > Language Options for *Selected Files*... > C++ tab > More Options... > Diagnostics tab)

Change certain ANSI C++ errors to warnings

Non-fatal ANSI C++ errors are downgraded to warnings.

Suppress all warnings

Suppresses all warning level messages from the compiler.

Quit building if warnings occur

Stop building when a warning occurs. Without the option, the build continues when warnings occur but stops when an error occurs.

Issue remarks

Issues remark level messages from the compiler, equivalent to mild warnings.

No “used before set” warnings

Does not issue warnings on local automatic variables used before their values are set.

No warnings for old for-loop scoping

If the new for-loop scoping is used, do NOT give a warning for programs which would have different behavior under the old rules.

Display message numbers

Displays the error message number in the diagnostic messages. These message number may be used in the **Suppress specific diagnostic** and **Change severity to...** text fields.

Display brief messages

Enables an error reporting mode in which a single line error message is produced. The original source line and pointer to the location of the error within that line is not displayed.

Don’t wrap diagnostic messages

Displays messages in single long lines, instead of wrapping them.

Maximum number of error mgs

Forces the compiler to abandon the compilation after this number of error messages, not including warning and remark level messages. The default limit is 100.

Suppress specific diagnostic

Suppresses diagnostic messages corresponding to the given error numbers. Multiple error numbers may be listed in a comma separated list. Use the Display message numbers check box to list error numbers in the diagnostic output.

Change severity to remark

Overrides the normal severity of the specified diagnostic message, making it a remark, whenever possible. Multiple error messages may be listed in a comma

separated list. Use the Display message numbers check box to list error numbers in the diagnostic output.

Change severity to warning

Overrides the normal severity of the specified diagnostic message, making it a warning, whenever possible. Multiple error numbers may be listed in a comma separated list. Use the Display message numbers check box to list error numbers in the diagnostic output.

Change severity to error

Overrides the normal severity of the specified diagnostic message, making it a fatal error. Multiple error number may be listed in a comma separated list. Use the Display message numbers check box to list the error numbers in the diagnostic output.

More C++ Options > Listing tab

(Builder: Project > Language Options for *Selected Files*... > C++ tab > More Options... > Listing tab)

Cross reference file

Generates a cross reference file corresponding to the source file. The file extension of cross reference files is **.xrf**.

Listing file

Generates a listing file corresponding to the source file. The file extension of a listing file is **.lis**.

Listing Directory

The directory in which to generate the cross reference and listing files.

Language Options > Ada tab

(Builder: Project > Language Options for *Selected Files*... > Ada tab)

The following are descriptions of the items in the **Ada** tab.

Main program name

Specifies the name of the main procedure for your Ada build (if .bld file is not labeled with the name).

Library directories

Allows additional library paths to be searched for libraries to be incorporated into the program. Equivalent to the **-L*dir*** command line option.

Elaboration only library directories

Allows additional elaboration only library paths to be incorporated into the program. Equivalent to the **-ep** build-time option to **adaopts** (**adaopts** is an Ada utility program; see the *Ada Language User's Guide*).

Ada83 analysis mode

Gives useful hints on converting Ada83 code to Ada95 code. It does not generate object code (strictly analysis mode). Same as **-ada83**.

Suppress all runtime checks

Suppresses all automatic run-time checking including numeric checking. This option is equivalent to using **Pragma Suppress** on all checks. Using this option reduces the size of the code. Same as **-no_check**.

Suppress numeric runtime checks

Suppresses two kinds of Numeric Checks for the entire compilation: **division_check** and **overflow_check**. The Ada95 LRM describes these checks. Using this option reduces the size of the code. Same as **-no_num_check**.

Generate cross reference

Generates a cross reference listing containing a line-numbered listing, followed by a cross reference table. The listing is written to **file.xlst**. Same as **-list/x**.

Generate text elaboration table

Generates an elaboration table listing in the **elab_table.txt** file.

Source listing (drop-down list)

Will generate source listing Always, Only if errors, or Never (default).

Always	-list/c
Only if errors	-list/e
Never	no listing no options thrown

Listing format (drop-down list)

Displays all source lines, all source lines numbered, and only error lines.

All source lines	default
All lines numbered	-list/p
Only error lines	-list/r

Page length/width

Allows you to format page length and width for the paginated source listing. Same as **-page/l** and **-page/w** respectively.

Diagnostics

Informs the Builder what to display in the progress window when building the application.

Suppress errors is the same as **-nomsg/e**.

Suppress warnings is the same as **-nomsg/w**.

Suppress informative messages is the same as **-nomsg/i**.

Suppress implementation dependent messages is the same as **-nomsg/d**.

This is a description of the Ada95 Library Displays:

Library info

Displays search path to libraries used by the application.

Registered units

Displays Unit Names and Unit descriptions of modules registered in the Ada library.

Registered sources

Displays source code path and names of modules registered in the Ada library.

Language Options > FORTRAN tab

(Builder: Project > Language Options for *Selected Files*... > FORTRAN tab)

FORTRAN version (drop-down list)

Standard

Interprets FORTRAN code in compliance with the ANSI FORTRAN standard.

F77

Interprets FORTRAN code for compatibility with AT&T's f77 compiler. Equivalent to the **-f77** command line options.

DoD

Enables DoD FORTRAN extensions. Equivalent to the **-dod** command line option.

Vax/VMS

Interprets code for compatibility with DEC's VAX/VMS FORTRAN compiler. This includes all Dod extensions. Equivalent to the **-vms** command line option.

Extended

Allows as many general purpose language extensions as possible.

Enable Debug Lines

Compiles lines starting with **d**, **D**, **x**, or **X**. The default treats them as comments. This option enables debugging statements. Equivalent to the **-d_line** command line option.

Namelist

Enables the IBM and VMS compatible **NAMELIST** extensions in FORTRAN. These extensions are already enabled in VMS compatibility mode. Equivalent to the **-namelist** command line option.

132 columns

Extends source to interpret columns 1 through 132 instead of only 1 through 72. Equivalent to the **-extend_source** command line option.

Implicit Undefined

Makes the default data type for undeclared variables as "undefined", equivalent to coding **IMPLICIT UNDEFINED(A-Z)** at the top of the source file. Equivalent to the **-u** command line option.

Case Sensitive

Does not convert uppercase user-supplied variables to lowercase. By default, FORTRAN is not case sensitive and all FORTRAN names are converted to lowercase. The compiler and library both assume this translation is performed. This option generally accesses variables defined in C as uppercase. However, when using this option, all FORTRAN keywords must be lowercase, making

the compiler incompatible with the ANSI FORTRAN-77 standard. Equivalent to the **-U** command line option.

Locals on Stack

Allocates local variables to registers or stacks, equivalent to coding **IMPLICIT AUTOMATIC (A-Z)** at the start of every subroutine or function. Programs compiled with this option are compliant with ANSI FORTRAN-77 and in some cases execute much more quickly. Equivalent to the **-nosave** command line option.

Check array bounds at runtime

Check that array subscripts are within the bounds of an array at runtime. Equivalent to the **-boundcheck** command line option.

One Trip Do Loops

Executes at least one iteration for every **DO** loop. By default, when the lower bound index of a **DO** loop is greater than the upper bound index, the compiler does not execute the **DO** loop for compatibility with the ANSI FORTRAN-77 standard. This option may be required for some older FORTRAN-66 programs to operate correctly. Equivalent to the **-onetrip** command line option.

VMS Common

Names **COMMON** blocks in the VMS style with a dollar sign appended. This option is enabled by default in VMS compatibility mode, but is also selected in F77 compatibility mode. Equivalent to the **-vms_common** command line option.

VMS Octal

Controls whether a double quotation mark is used for octal characters. If this is set, then the quotation mark is used for octal characters even in F77 and Extended modes. If this is not set, then the double quotation mark is an alternative to an apostrophe as a delimiter for character string constants. For example, **PRINT*,“sofa sofa”** prints **sofa sofa**. This is true even in VMS mode. The quotation mark is not allowed in Standard mode. Equivalent to the **-vms_octal** command line option.

2 Byte Integer

Sets the type for **INTEGER** to **INTEGER*2**. The default is **INTEGER*4**. Equivalent to the **-i2** command line option.

Hollerithblankpad

Pads hollerith constants on the right with blanks. The default, compatible with F77 mode, is that only the first byte of the hollerith is significant and the constant zero is padded on the right. Equivalent to the **-hollerith_blank_pad** command line option.

Missing Args Ok

Allows **CALL X(1,,2)**. Suppresses warning resulting from the missing argument. The compiler in either case passes a null value for the missing argument. Equivalent to the **-missing_args_ok** command line option.

Language Options > Pascal tab

(Builder: Project > Language Options for *Selected Files*... > Pascal tab)

Pascal version (drop-down list)

ISO Level 0
: Interprets Pascal code in compliance with the ISO Level 0 standard.

ISO Level 1
: Interprets Pascal code in compliance with the ISO Level 1 standard.

Extended
: Accepts all available Pascal extensions.

Big Set

Allocates all sets in the range **0..255**. Otherwise, sets are allocated in the range **0..31** for efficiency. ISO Level 0 and Level 1 Pascal default to Big Set, but Extended Pascal does not.

Case Sensitive

Specifies that Pascal is case sensitive. Extended Pascal defaults to case sensitive and ISO Level 0 and Level 1 Pascal does not. Equivalent to the **-X59** command line option.

Append score

Appends an underscore to the names of all functions and procedures. This prevents conflict with C routines that have the same name. However, this results in unresolved symbols because the Green Hills Pascal library does not expect to have this option set.

CPU Options dialog box

This is a complete description of the CPU specific option dialog boxes. MULTI displays only the CPU options dialog box that applies to the processor for which you are building your program.

Note: To set the processor family for your program, choose Project > Set Build Target For Project..., and pick a target.

For more detailed information on any of the options in these windows, please refer to the appropriate *Development Guide*.

i386/i486/Pentium dialog box

Processor (drop-down list)

Generates code optimized for the selected processor's instruction set.

Floating point processor (drop-down list)

Default

Generates code using the floating point capabilities of the selected processor or software floating point if the selected processor has no floating point support.

None

Rejects any use of floating point variables or constants in C, C++, or Pascal. Equivalent to the **-fnone** build-time option.

Software

Generate software floating point emulation code, regardless of the capabilities of the selected processor. Libraries built for software will also be used. Equivalent to the **-fsoft** build-time option.

The following are descriptions of the check boxes in the **i386/i486/Pentium** window.

fprecise

Stores all floating point calculations in memory to ensure precise truncation. Normally, all floating point operations on the 386/486/Pentium are done in extended precision. Without this option, calculations on the 386/486/Pentium are done at a different precision than on other architectures, which is sometimes undesirable. This option generates more predictable results, but makes your code larger and slower. Equivalent to the **-fprecise** command line option.

ffunctions

Enables the compiler to directly use the 387 hardware instructions for certain floating point functions instead of calling them in the library. Equivalent to the **-ffunctions** machine specific option.

manifest

The Builder predefines many symbols for compatibility with the SCO native C compiler. These symbols are known as manifest defines and all begin with **M_**. This option is only relevant for an SCO target system. Equivalent to the **-nomanifest** command line option.

The following options only apply to native Win32 compilation. Both fields are passed directly to the Microsoft linker:

Reserve

How much stack space you want allocated.

Commit

How much of that stack space you want unpaged.

The Microsoft linker documentation provides further information on these fields.

MC68000 dialog box

Processor (drop-down list)

68000

Generates code for the 68000 instruction set. Equivalent to the **-68000** machine specific option.

68010

Generates code for the 68010 instruction set. Equivalent to the **-68010** machine specific option.

68020

Generates code for the 68020 instruction set. Equivalent to the **-68020** machine specific option.

68030

Generates code for the 68030 instruction set. Equivalent to the **-68030** machine specific option.

68040

Generates code for the 68040 instruction set. Equivalent to the **-68040** machine specific option.

68LC040

Generates code for the 68LC040 instruction set. Equivalent to the **-68LC040** machine specific option.

68EC040

Generates code for the 68EC040 instruction set. Equivalent to the **-68EC040** machine specific option.

68060

Generates code for the 68060 instruction set. Equivalent to the **-68060** machine specific option.

68LC060

Generates code for the 68LC060 instruction set. Equivalent to the **-68LC060** machine specific option.

68EC060

Generates code for the 68EC060 instruction set. Equivalent to the **-68EC060** machine specific option.

6830x

Generates code for the 68000 instruction set used by the 68302 and 68306. Equivalent to the **-68302** machine specific option.

6833x

Generates code for the CPU32 instruction set used by the 68330, 68331, 68332, and 68F333. Equivalent to the **-68331** machine specific option.

68340

Generates code for the CPU32 instruction set used by the 68340. Equivalent to the **-68340** machine specific option.

68360

Generates code for the CPU32+ instruction set used by the 68360. Equivalent to the **-68360** command line option.

MCF510x

Generates code for the ColdFire 5100 series instruction set. Equivalent to the **-cf5102** command line option.

MCF5202

MCF5203

MCF5204

MCF5206E

Generates code for the ColdFire 5200 series instruction set. Equivalent to the **-cf5202**, **-cf5203**, **-cf5204**, **-cf5206, -cf206e** machine specific options.

MCF5307

Generates code for the ColdFire 5300 series instruction set. Equivalent to the **-cf5307** machine specific option.

Floating point processor (drop-down list)

Default

Generates code using the floating point capabilities of the selected processor or software floating point if the selected processor has no floating point support.

None

Rejects any use of floating point variables or constants in C, C++, or Pascal. Equivalent to the -**fnone** build-time option.

Software

Generate software floating point emulation code, regardless of the capabilities of the selected processor. Libraries built for software will also be used. Equivalent to the **-fsoft** build-time option.

68881

Generates code for the 68881 floating point processor. Equivalent to the **-68881** machine specific option.

68882

Generates code for the 68882 floating point processor. Equivalent to the **-68882** machine specific option.

Position independent code (drop-down list)

Absolute

Generates absolutely addressed (position dependent) code. Equivalent to the **-nopic** command line option.

16 bit pc-relative

Generates position independent code for the code and data sections of the program. The position offsets are 16 bits (+/-

32KB), and are relative to the program counter. Equivalent to the **-pic16** PIC option.

32 bit pc-relative

Generates position independent code for the code and data sections of the program. The position offsets are 32 bits, and are relative to the program counter. Equivalent to the **-pic32** PIC option.

Position independent data (drop-down list)

Absolute

Generates absolute addressed (position dependent) data. Equivalent to the **-nopid** PID option.

16 bit pc-relative

Generates position independent data for the data sections of the program. The position offsets are 16 bits (+/- 32KB), and are relative to the program counter. Equivalent to the **-pid16** option.

16 bit a2-relative

Generates position independent data for the data sections of the program. The position offsets are 16 bits (+/- 32KB), and are relative to the register **a2**. Equivalent to the **-pid16=a2** option.

16 bit a3-relative

Generates position independent data for the data sections of the program. The position offsets are 16 bits (+/- 32KB), and are relative to the register **a3**. Equivalent to the **-pid16=a3** option.

16 bit a4-relative

Generates position independent data for the data sections of the program. The position offsets are 16 bits (+/- 32KB), and are relative to the register **a4**. Equivalent to the **-pid16=a4** option.

16 bit a5-relative

Generates position independent data for the data sections of the program. The position offsets are 16 bits (+/- 32KB), and are relative to the register **a5**. Equivalent to the **-pid16=a5** option.

16 bit a6-relative

Generates position independent data for the data sections of the program. The position offsets are 16 bits (+/- 32KB), and are relative to the register **a6**. Equivalent to the **-pid16=a6** option.

32 bit pc-relative
Generates position independent data for the data sections of the program. The position offsets are 32 bits, and are relative to the program counter. Equivalent to the **-pid32** option.

32 bit a2-relative
Generates position independent data for the data sections of the program. The position offsets are 32 bits, and are relative to the register **a2**. Equivalent to the **-pid32=a2** option.

32 bit a3-relative
Generates position independent data for the data sections of the program. The position offsets are 32 bits, and are relative to the register **a3**. Equivalent to the **-pid32=a3** option.

32 bit a4-relative
Generates position independent data for the data sections of the program. The position offsets are 32 bits, and are relative to the register **a4**. Equivalent to the **-pid32=a4** option.

32 bit a5-relative
Generates position independent data for the data sections of the program. The position offsets are 32 bits, and are relative to the register **a5**. Equivalent to the **-pid32=a5** option.

32 bit a6-relative
Generates position independent data for the data sections of the program. The position offsets are 32 bits, and are relative to the register **a6**. Equivalent to the **-pid32=a6** option.

The following are descriptions of the check boxes in the **MC-68000** window.

Use built-in fp funcs
Uses built-in floating point instructions rather than calling library functions such as **fabs()**, **sqrt()**, and **sin()**. Equivalent to the **-ffunctions** machine specific option.

Insert extra Fpnops
Appends one **FPNOP** instruction after every 68881 instruction, other than **FMOVE**, unless immediately followed by a 68881 instruction. This ensures that interrupts are taken at the correct position. Equivalent to the **-ffpnop** machine specific option.

Return fp in d0/d1
Returns floating point numbers from functions in the registers **d0** and **d1** instead of **fp0**. Equivalent to the **-freturnd0** machine specific option.

Truncate fp expressions

In 68881/68882 and 68040 mode, this stores all single and double precision floating point variables and values in memory to ensure precise truncation. Without this option, variables and intermediate values are often located in the internal 80-bit floating point registers, resulting in additional precision. This produces results different from other architectures that truncate all results to 32 or 64 bits. Equivalent to the **-fprecise** machine specific option.

Enable 68851 support

Enables use of a 68851 memory management unit. This option only affects the assembler. Equivalent to the **-68851** machine specific option.

Use DS for uninit vars

This is currently unsupported. For zero initialized variables and assembly output, the **DS** directive is used instead of **DCB**. This results in much smaller output from the assembler but does not explicitly initialize the variables to zero. It is then your responsibility to make sure the variables are initialized to zero.

Portable assembly code

For assembly output, use constant directives rather than actual 68K instructions. The compiler outputs assembly code which is portable to many more assemblers than the standard assembly code output. Equivalent to the **-preassemble** machine specific option.

Large switch statements

Allows large switch statements by forcing the compiler to use a 32-bit offset, which works regardless of the destination label. The default is a 16-bit offset which is smaller and faster. However, it fails if a label is too far away. Equivalent to the **-bigswitch** FORTRAN option.

Minimum structure alignment

Does not round up the size of structs containing one byte of data to the default minimum alignment value. These structs will be one byte in size and may be linked at an odd-numbered address with the option.

Pop stack args often

Subroutine calls require a substantial amount of code in many programs. Each time a subroutine is called the arguments are pushed on the stack, the subroutine is called, and when the subroutine returns the arguments are removed from the stack by adding to the stack pointer. In many programs a substantial savings in code size is realized by the Green Hills compiler by not adding to the stack pointer after each call.

Instead, the total amount of space that needs to be removed is accumulated until some occurrence, such as a branch, forces the

compiler to adjust the stack. The optimization may cause a program to use more stack space than it otherwise would have.

This option forces a stack adjustment after each subroutine call. This stops stack frames from growing too large at the expense of generating more code.

V800 dialog box

This window applies to the most up-to-date V800 toolset which uses ELF as the default format and version 1.8.9 compilers. To use the previous generation of tools and COFF or ELF format, either use the V805/V810/V820/V830 window or the V850/V851 window.

To change the target processor to a different member of the V800 series, change the entry in the Target box in the main Builder window. Do not change the processor field in this window.

Processor (drop-down list)

Generates code for the selected processor's instruction set. Equivalent to executing the corresponding processor specific compiler driver from the command line.

Floating point processor (drop-down list)

Default

Generates code using the floating point capabilities of the selected processor.

None

Rejects any use of floating point variables or constants in C, C++, or Pascal. Equivalent to the **-fnone** build-time option.

V850 tiny data area (drop-down list)

Allocates a small area of Tiny Data Area (TDA) memory to hold small data objects and reference objects in that area using a base pointer register (r30) and short load and store instructions. The total size of the Tiny Data Area is limited to 4K. Please refer to the "Tiny Data Area Optimization" chapter in the *Embedded V800 Development Guide* for more information.

You need to specify which TDA model to enable. Because TDA is enabled only through #pragma by default, no objects are placed in the TDA.

None
No TDA. The compiler uses the TDA base register r30 as a normal temporary register.

Single
Enables the Single TDA model.

Multiple
Enables the Multiple TDA model.

The following are descriptions of the check boxes and text fields in the V800 window:

Reserve r2 for the user
The compiler reserves r2 for the user. This is the default. If the box is set to "-", the compiler uses r2 as a temporary register. Equivalent to the **-reserve_r2** command line option.

Reserve r5 for the user
The compiler reserves r5 for the user. This is the default. If the box is set to "-", the compiler uses r5 as a temporary register. Equivalent to the **-reserve_r5** command line option.

Reserve r15-r24 for the user (22 register mode)
Generates code in 22 register mode. Default is 32 register mode. Equivalent to the **-cpu=v800_22** V800 specific option.

Reserve r17-r22 for the user (26 register mode)
Generates code in 26 register mode. Default is 32 register mode. Equivalent to the **-cpu=v800_26** V800 specific option.

Constant value 255 is in r20
The compiler assumes that r20 contains the value 255 and uses r20 instead of 255 during code generation. r20 should be initialized in the startup code. Equivalent to the **-r20has255** command line option.

Constant value 255 is in r20 and 65535 is in r21
The compiler assumes that r20 contains the value 255 and r21 contains the value 65535. The compiler will use r20 and r21 instead of 255 and 65535 during code generation. r20 and r21 should be initialized in the startup code. Equivalent to the **-r21has65535** command line option.

Position independent code
Generates position independent code. Equivalent to the **-pic** PIC option.

Position independent data
Generates position independent data. Equivalent to the **-pid** PID option.

Far function calls

Generates register-indirect calls (“far”-calls) for user functions. The default is not to generate far calls, but to use PC-relative calls. Equivalent to the **-farcalls** command line option.

Inline prologue

Forces the compiler to generate function prologue and epilogue code inline. The alternative is to call routines in the Green Hills libraries to save and restore registers and allocate stack space for locals. This option is on by default for unoptimized code, or when not optimizing for small code size.

Do not use V850E callt instruction

Prevents the compiler from generating the CALLT instruction on the V850E. Also causes the linker to use a different set of libraries which also do not use the CALLT instruction.

Small Data or Zero Data threshold

Specifies a size in bytes to determine which data objects appear in the Small or Zero Data Areas. By default, objects less than 8 bytes are placed in Small Data Area (i.e., the default small data area threshold is 8), and no objects are placed in Zero Data Area (i.e, the default zero data area threshold is 0). Equivalent to the **-sda=** and **-zda=** special data area options.

See the *Development Guide* for more information on the SDA and ZDA optimizations.

Put variables smaller than threshold into (drop-down list)

Normal Data

Puts variables smaller than threshold into the Data Area.

Small Data

Allocates an area of memory to hold data objects smaller than the Small Data Threshold and references objects in that area using r4 as the base pointer register. Equivalent to the **-sda** Small Data Area option.

Zero Data

Allocates an area of memory to hold data objects smaller than the Zero Data Threshold and references objects in that area using r0 as the base pointer register. This improves program size and speed because addressing an object via the Small/Zero data area base register uses fewer instructions. The total size of the Small/Zero data area is limited to 64k; large applications may not be able to

take advantage of this feature. Equivalent to the **-zda** special data area option.

i960 dialog box

Processor (drop-down list)

Generates code for the selected processor's instruction set. Equivalent to the **-cpu=960** machine-specific option.

Floating point processor (drop-down list)

Default

Generates code using the floating point capabilities of the selected processor or software floating point if the selected processor has no floating point support.

None

Rejects any use of floating point variables or constants in C, C++, or Pascal. Equivalent to the **-fnone** build-time option.

Software

Generates software floating point emulation code, regardless of the capabilities of the selected processor. Libraries built for software will also be used. Equivalent to the **-fsoft** build-time option.

The following is a description of the items in the i960 window.

Big-Endian

Generates code with big endian byte order. The most significant byte of an integer appears at the lowest address. Equivalent to the **-bigendian** command line option.

Position Independent Code

Generates position independent code. Equivalent to the **-pic** PIC option.

Position Independent Data

Generates position independent data. Equivalent to the **-pid** PID option.

Small Data Area

Allocates a small area of memory to hold small data objects and references objects in that area using a base pointer register. This improves program size and speed because addressing an object via the small data area base register uses fewer instructions. Equivalent to the **-sda** Small Data Area option.

Small Data Area Threshold

Specifies a size in bytes to determine which data objects appear in the Small Data Area. By default, objects four bytes or less are placed in the Small Data Area. Equivalent to the **-sda=** special data area option.

See the i960 *Development Guide* for more information about the Small Data Area optimization.

Alpha dialog box

Floating point processor (drop-down list)

Default

Generates code using the floating point capabilities of the Alpha processor.

None

Rejects any use of floating point variables or constants in C, C++, or Pascal. Equivalent to the **-fnone** build-time option.

ARM dialog box

Processor (drop-down list)

Generates code for the selected processor's instruction set. Equivalent to setting the corresponding **-cpu** flag on the command line.

Floating point processor (drop-down list)

Default

Generates code using the floating point capabilities of the selected processor.

None

Rejects any use of floating point variables or constants in C, C++, or Pascal. Equivalent to the **-fnone** build-time option.

Software

Generate software floating point emulation code, regardless of the capabilities of the selected processor. Libraries built for software will also be used. Equivalent to the **-fsoft** build-time option.

Fpal0

Generate code for the ARM Fpal0 hardware floating point unit. Equivalent to the **:arm_fputype=fpa10** command line option.

The following describes the items in the ARM window:

Big endian

Generates code with big endian byte order. The most significant byte of an integer appears at the lowest address. Equivalent to the **-bigendian** command line option.

Thumb code

Generate code for 16-bit instruction Thumb mode. Equivalent to the **-thumb** command line option.

Thumb libraries

Link with runtime libraries built for Thumb mode (default is ARM libraries).

FR20 dialog box

Floating point processor (drop-down list)

Default

Generates code using the floating point capabilities of the FR processor.

None

Rejects any use of floating point variables or constants in C, C++, or Pascal. Equivalent to the **-fnone** build-time option.

MCore dialog box

Processor (drop-down list)

Generates code for the selected processor's instruction set.

Floating point processor (drop-down list)

Default

Generates code using the floating point capabilities of the selected processor.

None

Rejects any use of floating point variables or constants in C, C++, or Pascal. Equivalent to the **-fnone** build-time option.

Software

Generates software floating point emulation code, regardless of the capabilities of the selected processor. Libraries built for software will also be used. Equivalent to the **-fsoft** build-time option.

Single Precision Hardware

Causes the compiler to use hardware instructions to do some single-precision floating point operations for the CPU types that support it.

MIPS dialog box

Processor (drop-down list)

Generates code for the selected processor's instruction set.

Floating point processor (drop-down list)

Default

Generates code using the floating point capabilities of the selected processor or software floating point code if the selected processor has no floating point support.

None

Rejects any use of floating point variables or constants in C, C++, or Pascal. Equivalent to the **-fnone** build-time option.

Software

Generates software floating point emulation code, regardless of the capabilities of the selected processor. Libraries built for software will also be used. Equivalent to the **-fsoft** build-time option.

Hardware Single

Uses hardware floating point for single precision, but software floating point for double precision (when this mode is supported in the selected processor). The compiler uses different libraries instead of the default. Equivalent to the **-fsingle** build-time option.

Calling sequence (drop-down list)

RH32 always uses the Embedded Calling Sequence, which is the default. Therefore this item should be kept as "Default" for RH32 (equivalent to the **-embedded_calling_sequence** command line option). Selecting "Workstation" (equivalent to the **-workstation_calling_sequence** command line option) would have undefined effect on the code generation.

RH32 FPU (drop-down list)

Select for which Floating Point Unit the floating point instructions are generated. When the FPU number is *n*, the compiler will append *n* to every

floating point instructions generated when appropriate. The default FPU number is 0. Equivalent to the **-fpu=n** option.

The following are descriptions of the items in the MIPS window.

Position Independent Code
Generates position independent code. Equivalent to the **-pic** PIC option.

Position Independent Data
Generates position independent data. Equivalent to the **-pid** PID option.

PIC Compatible Code
Generates code that does not use the PIC base register. Code compiled with this option is absolutely addressed, but can be safely linked with code compiled with the position independent code or position independent data options.

MIPS Assembler Compatible Output
Produces assembly language code for the MIPS native assembler. By default, code is produced for the Green Hills assembler.

Little-Endian
Generates code with little endian byte order. The least significant byte of an integer appears at the lowest address. This is the default with native compilers on the DEC station and is also used for embedded development. Equivalent to the **-littleendian** command line option.

MIPS-16 Instruction Set
Enables the MIPS-16 ISA. Equivalent to the **-mips16** command line option.

MIPS-16 Library
Link in the MIPS-16-specific runtime libraries (prebuilt with MIPS-16 enabled) by default, when the MIPS-16 instruction set is enabled. If the MIPS-16 ISA is not enabled, clicking this button will have no effect. Equivalent to the **-mips16_lib** command line option.

64 bit integers
Enables the 64-bit mode. In this mode, all registers are 64-bit wide, and 64-bit integers and arithmetical are supported (via the **long long** type). However, pointers and addresses are still 32 bits. This option is not available with all processors. Equivalent to the **-64bit** command line option.

Far Function Calls
Enables the far function call mode. All functions are called via the **jalr** instruction instead of the **jal** instruction by first loading the address of

the function into a temporary register. This allows the functions called to be in any address within the range of 32 bits. Equivalent to the **-farcalls** command line option.

Inline Prologue

Always generates the inlined prologue/epilogue code that saves and restores the permanent registers in/from the stack of a function. If this option is not turned on, then the compiler will choose for each function to generate the prologue/epilogue code either inlined or offlined (when in Software Floating Point mode, and not with -ga nor -G), depending on which is better for code size when the # of registers to be saved/restored are more than 2.

Note: An offline prologue/epilogue is a function call to a library routine **__savegN** that saves registers $16 to $N, or **__restgN** that restored registers $16 up to $N, respectively. These routines exist in indarch.a.

Multiple Near Code Regions (MIPS-X only)

Generates code in 'multiple near code region mode'. This is used to create multiple code regions in memory. Code in each region is accessible via an offset from a base register whose initial value is determined during link time by specifying the **-initreg** linker option. This feature is available for MIPS-X only. See your *Development Guide* for more information.

Small Data or Zero Data Threshold

Specifies a size in bytes to determine which data objects appear in the Small or Zero Data Areas. By default, objects less than 8 bytes are placed in Small Data Area, and no objects are placed in Zero Data Area. Equivalent to the **-sda=** and **-zda=** special data area options.See the MIPS *Development Guide* for more information about the Small Data Area optimization.

Put variables smaller than threshold size into (drop-down list)

Normal Data

Puts variables smaller than threshold into the Data Area.

Small Data

Allocates an area of memory to hold data objects smaller than the Small Data Threshold and references objects in that area using r4 as the base pointer register. Equivalent to the **-sda** Small Data Area option.

Zero Data

Allocates an area of memory to hold data objects smaller than the Zero Data Threshold and references objects in that area using r0

as the base pointer register. This improves program size and speed because addressing an object via the Small/Zero Data Area base register uses fewer instructions. The total size of the Small/Zero Data Area is limited to 64k; large applications may not be able to take advantage of this feature. Equivalent to the **-zda** special data area option.

nCPU dialog box

Floating point processor (drop-down list)

Default

Generates code using the floating point capabilities of the selected processor or software floating point if the selected processor has no floating point support.

None

Rejects any use of floating point variables or constants in C, C++, or Pascal. Equivalent to the **-fnone** build-time option.

Software

Generates software floating point emulation code, regardless of the capabilities of the selected processor. Libraries built for software will also be used. Equivalent to the **-fsoft** build-time option.

The following are descriptions of the check boxes in the nCPU window.

Far function calls

This causes the compiler to generate function calls through a register; this allows for functions to be located at any distance from the caller. Without this, extremely large programs or programs with discontinuous text sections may not link if the range of the call instruction is exceeded.

Small Data Area Threshold

Specifies a size in bytes to determine which data objects appear in the Small Data Area. By default, objects four bytes or less are placed in the Small Data Area. Equivalent to the **-sda=** special data area option.

See the *Development Guide* for more information about the Small Data Area optimization.

Put variables smaller than threshold size into (drop-down list)

Normal Data

Puts variables smaller than threshold into the Data Area.

Small Data

Allocates an area of memory to hold data objects smaller than the Small Data Threshold and references objects in that area using r4 as the base pointer register. Equivalent to the **-sda** Small Data Area option.

Zero Data

Allocates an area of memory to hold data objects smaller than the Zero Data Threshold and references objects in that area using r0 as the base pointer register. This improves program size and speed because addressing an object via the Small/Zero data area base register uses fewer instructions. The total size of the Small/Zero data area is limited to 64k; large applications may not be able to take advantage of this feature. Equivalent to the **-zda** special data area option.

NDR dialog box

Floating point processor (drop-down list)

Default

Generates code using the floating point capabilities of the selected processor or software floating point if the selected processor has no floating point support.

None

Rejects any use of floating point variables or constants in C, C++, or Pascal. Equivalent to the **-fnone** build-time option.

The following describes the items in the NDR window.

Position independent code

Generates position independent code. Equivalent to the **-pic** PIC option.

Position independent data

Generates position independent data. Equivalent to the **-pid** PID option.

Zero data area

Enables the ZDA optimization. Puts variables smaller than the threshold in the Zero Data Area. Equivalent to the **-zda=** special data area option.

Zero data area threshold

Specifies a size in bytes to determine which data objects appear in the Zero Data Area. Equivalent to the **-zda=special** data area option.

PowerPC dialog box

Processor (drop-down list)

Generates code for the selected processor's instruction set.

Floating point processor (drop-down list)

Default

Generates code using the floating point capabilities of the selected processor or software floating point if the selected processor has no floating point support.

None

Rejects any use of floating point variables or constants in C, C++, or Pascal. Equivalent to the **-fnone** build-time option.

The following are descriptions of the check boxes in the PowerPC window.

Constant Data Section

Places all string literals, floating point constants, and initialized variables declared const in C and C++ in a separate section, rather than the section containing other initialized data.

This option is on by default. When disabled, constant data will be placed in the **.data** or **.sdata** section. The user should take care when using this option and interfacing with the provided libraries. The libraries place constant data in **.rodata** sections, and problems can come up due to the inconsistency, particularly if the user turns on Position Independent Code or Data.

Position Independent Code

Generates position independent code. Equivalent to the **-pic** PIC option.

Position Independent Data

Generates position independent data. Equivalent to the **-pid** PID option.

Far Function Calls

Uses the PowerPC instructions **mtlr** followed by **blrl** instead of **bl** for all calls so that a full 32-bit displacement may be used. This is not supported with position independent code. Equivalent to the **-farcalls** command line option.

Inline Prologue

Normally, the compiler chooses the most efficient function prologue and epilogue, depending on the optimization settings. This option prevents the compiler from calling off to library routines for this purpose; inline

code sequences will be used instead. This option may adversely impact the size of the generated code, so it should only be used when it is necessary (for example, when the routines may not exist in memory yet).

Little Endian

Specifies code generation for a little endian system. Equivalent to the **-littleendian** command line option.

Label at End of Function

Specifies that a global label of the form **__ghs_eofn_funcname** will be placed at the end of every function. This can be useful for computing the size of functions based on global symbols.

Truncate single-precision fp on RSC:

This option only affects the Power architecture **-cpu=rsc**. Unlike PowerPC, the closely related Power Architecture/RS6000 does not have arithmetic instructions that produce a single precision result. As a result, code which depends on the exact precision of single precision quantities may not execute correctly. This option causes the compiler to truncate the result to a single precision quantity after each single precision arithmetic operation; this will cause the code to have the desired behavior in these cases. This option is meaningless for non-RS6000 members of the Power/PowerPC family.

Small Data or Zero Data Threshold

Specifies a size in bytes to determine which data objects appear in the Small or Zero Data Areas. By default, objects less than 8 bytes are placed in Small Data Area (i.e. the default small data area threshold is 8), and no objects are placed in Zero Data Area (i.e. the default zero data areathresholdis0).Equivalenttothe-**sda=**and-**zda=**specialdataareaoptions.

See the *Development Guide* for more information on the SDA and ZDA optimizations.

Put variables smaller than threshold size into (drop-down list)

Species the XDA section for variables smaller than the specified threshold. If “Normal Data” is selected, all variables will be placed in normal data rather than the SDA or ZDA sections.

SH dialog box

Processor (drop-down list)

Generates code for the selected processor’s instruction set.

Floating point processor (drop-down list)

Default

Generates code using the floating point capabilities of the selected processor or software floating point if the selected processor has no floating point support..

None

Rejects any use of floating point variables or constants in C, C++, or Pascal. Equivalent to the **-fnone** build-time option.

Software

Generate software floating point emulation code, regardless of the capabilities of the selected processor. Libraries built for software will also be used. Equivalent to the **-fsoft** build-time option.

The following describes the items in the SH window.

Position Independent Code

Generates position independent code. Equivalent to the **-pic** PIC option.

Position Independent Data

Generates position independent data. Equivalent to the **-pid** PID option.

Little Endian

Specifies code generation for a little endian system. Equivalent to the **-littleendian** command line option.

All Floating Point is Single Precision

This option will cause "double" to be interpreted as "float" so no 64-bit instructions will be required for floating point operations. Equivalent to the **-floatsingle** command line option.

Disable use of MACH, MACL, and GBR by compiler

Prevents the compiler from using the MACH, MACL, or GBR registers as general purpose, permanent registers.

Small Data Area

Allocates a small area of memory to hold small data objects and references objects in that area using a base pointer register. This may improve program size and speed because addressing an object via the Small Data Area base register sometimes uses fewer instructions. Equivalent to the **-sda** small data area option.

Small Data Area Threshold

Specifies a size in bytes to determine which data objects appear in the Small Data Area. Equivalent to the **-sda=** special data area option.

SPARC dialog box

Processor (drop-down list)

Generates code for the selected processor's instruction set.

Floating point processor (drop-down list)

Default

Generates code using the floating point capabilities of the selected processor or software floating point if the selected processor has no floating point support.

None

Rejects any use of floating point variables or constants in C, C++, or Pascal. Equivalent to the **-fnone** build-time option.

Software

Generates software floating point emulation code, regardless of the capability of the selected processor. Libraries built for software will also be used. Equivalent to the **-fsoft** build-time option.

The following are descriptions of the check boxes in the Sparc window.

pic (small offset)

Generates System V.4 style Position Independent Code with 16-bit offsets. Code generated with this option is placed into a shared object. Equivalent to the **-pic** PIC option.

PIC (large offset)

Generates System V.4 style Position Independent Code with 32-bit offsets. Code generated with this option is placed into a shared object. The larger offset degrades program size and speed, but increases the limit on the number of external symbols appearing in a shared object. All modules in a single shared object are compiled with the same type of offset. Equivalent to the **-PIC** command line option.

Assume Double Alignment

By default, 4-byte loads and stores access all 8-byte objects in memory to avoid any errors caused by using an 8-byte load on an address which is a multiple of four, but not a multiple of eight. This option uses 8-byte loads and stores to access 8-byte objects. This improves program size and speed, but requires all 8-byte objects to align properly. Equivalent to the **-dalign** command line option.

Reserve Registers g5, g6, g7 for User

This option will keep the compiler from using the %g5, %g6 or %g7 registers as general purpose, permanent registers.

Small Data Area

Allocates a small area of memory to hold small data objects and references objects in that area using a base pointer register. This improves program size and speed because addressing an object via the Small Data Area base register uses fewer instructions. The total size of the Small Data Area is limited to 8K; therefore, large applications may not be able to take advantage of this feature. Equivalent to the **-sda** command line option.

ST100 dialog box

Floating point processor (drop-down list)

Default

Generates code using the floating point capabilities of the selected processor or software floating point if the selected processor has no floating point support.

None

Rejects any use of floating point variables or constants in C, C++, or Pascal. Equivalent to the **-fnone** build-time option.

The following describes the items in the ST100 dialog box.

Position Independent Code

Generates position independent code. Equivalent to the **-pic** PIC option.

Gp 16 Mode

Causes code to be generated in the gp16 instruction set of the ST100. This provides smaller overall code size, at the expense of code speed

Gp 16 Libraries

Causes your application to be linked with standard libraries that were built in gp16 mode (see above). The gp16 libraries are smaller in size, but may execute more slowly.

Small Data Area Threshold

Specifies a size in bytes to determine which data objects appear in the Small Data Area. Equivalent to the **-sda=** small data area build-time option.

Tiny Data Area Threshold

Specifies a size in bytes to determine which data objects appear in the Tiny Data Area. Equivalent to the **-tda=** tiny data area build-time option.

StarCore dialog box

Floating point processor (drop-down list)

Default

Generates code using the floating point capabilities of the selected processor or software floating point if the selected processor has no floating point support.

None

Rejects any use of floating point variables or constants in C, C++, or Pascal. Equivalent to the **-fnone** build-time option.

The following describes the items in the StarCore window.

Big Endian

Specifies code generation for a big endian system. Equivalent to the **-b** command line option.

Far function call

Treats all function calls as far calls. Equivalent to the **-farcalls** command line option.

Align functions to 16-byte boundaries

Causes all functions to be aligned on 16-byte boundaries.

Do not allocate to d8-d15

Prevents the compiler from allocating variables to the high data registers, d8 through d15.

Do not allocate to r8-r15

Prevents the compiler from allocating variables to the high address registers, r8 through r15.

Small Data or Zero Data threshold

Specifies a size in bytes to determine which data objects appear in the Small or Zero Data Areas. By default, objects less that 8 bytes are placed in the Small Data Area (i.e the default small data area threshold is 8), and no objects are placed in the Zero Data Area (i.e the default zero data area threshold is 0). Equivalent to the **-sda=** and **-zda=** special data area options.

See the *Development Guide* for more information on the SDA and ZDA optimizations.

Put variables smaller than threshold size into (drop-down list)

Normal Data

Puts variables smaller than threshold into the data area.

Small Data

Allocates an area of memory to hold data objects smaller than the Small Data threshold and references objects in that area using r4 as the base pointer register. Equivalent to the **-sda** Small Data Area option.

Zero Data

Allocates an area of memory to hold data objects smaller than the Zero Data threshold and references objects in that area using r0 as the base pointer register. This improves program size and speed because addressing an object via the Small/Zero Data Area base register uses fewer instructions. The total size of the Small/Zero Data Area is limited to 64k; large applications may not be able to take advantage of this feature. Equivalent to the **-zda** special data area option.

TriCore dialog box

Processor (drop-down list)

AUDO-lite

For use with AUDO-lite boards.

AUDO-1

For use with AUDO-1 boards.

Rider A

For use with the TriCore Rider A Evaluation Board.

Rider B

For use with the TriCore Rider B Evaluation Board.

Floating point processor (drop-down list)

Default

Generates code using the floating point capabilities of the selected processor or software floating point if the selected processor has no floating point support..

None

Rejects any use of floating point variables or constants in C, C++, or Pascal. Equivalent to the **-fnone** build-time option.

The following describes the item boxes in the TriCore window.

Far Function Calls

Treats all function calls as far calls.

A function call is normally performed with a CALL instruction, which takes a 24-bit PC-relative displacement. In large programs with functions placed throughout memory, you can make functions unreachable with the CALL instruction. This will result in an error at link time.

To avoid this problem, the compiler provides support for Far Function Calls. In place of the CALL instruction, the compiler places the address of the called function into a register and uses a CALLI instruction to perform an indirect call. This method can reach a function anywhere in the address space of the processor, at the expense of slightly larger code.

Small Data or Zero Data Threshold

Specifies a size in bytes to determine which data objects appear in the Small or Zero Data Areas. By default, objects less that 8 bytes are placed in Small Data Area (i.e the default small data area threshold is 8), and no objects are placed in Zero Data Area (i.e the default zero data area threshold is 0). Equivalent to the **-sda=** and **-zda=** special data area options.

See the *Development Guide* for more information on the SDA and ZDA optimizations.

Put variables smaller than threshold size into (drop-down list)

Normal data

Puts variables smaller than threshold into the data area.

Small data

Allocates an area of memory to hold data objects smaller than the Small Data threshold and references objects in that area using r4 as the base pointer register. Equivalent to the **-sda** Small Data Area option.

Zero data

Allocates an area of memory to hold data objects smaller than the Zero Data threshold and references objects in that area using r0 as the base pointer register. This improves program size and speed because addressing an object via the Small/Zero Data Area base register uses fewer instructions. The total size of the Small/Zero Data Area is limited to 64k; large applications may not be able to take advantage of this feature. Equivalent to the **-zda** special data area option.

Toolchain Options dialog box

This is a complete description of the toolchain specific option dialog boxes. MULTI displays only the toolchain options window that applies to the toolchain with which you are building your program. For more detailed information on any of the options in these windows, please refer to the appropriate *Development Guide*.

Toolchain Options > Linker tab

(Builder: Project > Toolchain Options for *Selected Files*... > Linker tab)

Linker section overlap (drop-down list)
Allows sections to overlap in the final layout without warning or error.

Disable warnings
Disables warnings from the linker.

Generate checksum
Generates a 4-byte checksum at the end of every section, equivalent to the **-checksum** option. See the *Development Guide* for more information.

Allow undefined symbols
Allows a link to complete successfully with undefined symbols rather than producing an error. Equivalent to the **-undefined** command line option.

Re-scan libraries
Continually re-scans the list of libraries to resolve dependencies if at the end of a pass through the libraries some symbols remain unresolved. This continues until either all symbols are resolved or no symbols were resolved in the last pass. See the linker section in the *Development Guide* for more information.

Relocatable program
Generates an object file which is suitable for input into another linker run, a dynamic loader, or an executable. Equivalent to linking with the **-relprog** option.

No INTEGRITY shared objs
INTEGRITY specific: do not use shared objects (libraries are statically linked into each AddressSpace).

C parameter checking
Causes the C compiler to output special symbols which represent every function definition and every function call. The Green Hills linker (lx or elxr) will recognize these symbols and give an error if a function is

called inconsistently with its definition. Equivalent to the build-time option **-parameter_check**.

Full C parameter checking
This is a superset of "C parameter checking" above. It causes the linker to give an error if a function is called but there is no parameter checking information associated with the definition. This option cannot be used unless all libraries and objects are compiled with some form of C parameter checking enabled. Furthermore, any functions defined in assembly language will need to have parameter checking information added manually. Note that libraries provided by Green Hills are not compiled with parameter checking enabled except on certain targets. Equivalent to the build-time option **-full_parameter_check**.

Undefined symbols
Identify these symbols as undefined in the symbol table. This is generally used to force the loading of a library symbol that otherwise might not be loaded. Equivalent to the **-u** *sym* command line option.

Defined symbols
The format is *sym=val*. Defines a symbol '*sym*' with value '*val*'.

Abs objects to link against
Equivalent to linking with **-A** <*name of object*>

General linker options
These options are passed through to the linker.

Generate map file
Generate a link map file showing where symbols are located. See the linker chapter in the *Development Guide* for more information. Equivalent to the **-map** command line option.

Add cross reference
Adds cross reference information to the link map. See the linker chapter in the *Development Guide* for more information. Equivalent to the **-Mx** command line option.

Sort symbols by address
Sorts symbols in the link map by their address. See the linker chapter in the *Development Guide* for more information. Equivalent to the **-Mn** command line option.

Wide format
Prints the link map in wide format. See the linker chapter in the *Development Guide* for more information. Equivalent to the **-Mw** command line option.

Map file name
Specifies the filename for the map file (optional). Equivalent to the **-map=*name*** command line option**.**

Map file directory
Specifies the directory where the map file will be generated (optional). Equivalent to the **-map=*name*** command line option.

Toolchain Options > Assembler tab

(Builder: Project > Toolchain Options for *Selected Files*... > Assembler tab)

Disable warnings
Disable warnings from the assembler.

Error for undefined sym
Give an error for undefined symbols. Normally undefined symbols are silently converted into external references.

General assembler options
These options are passed through to the assembler. Equivalent to using the **-asm=** command line option.

Generate listing file
Generates an assembly listing file. See the assembler chapter in the *Development Guide* for more information. Equivalent to the **-list** command line option.

Add cross reference
Adds cross reference information to the assembly listing file. See the assembler chapter in the *Development Guide* for more information. Equivalent to the **-ref** command line option.

Do not expand macros
Do not expand macros in the assembly listing file. See the assembler chapter in the *Development Guide* for more information. Equivalent to the **-nogen** command line option.

Listing file name
Specifies the filename for the assembly listing file (optional). Equivalent to using the **-list=*name*** command line option.

Listing file directory
Specifies the directory where the assembly listing file will be generated (optional). Equivalent to using the **-list=*name*** command line option.

68000 Toolchain Options > Linker tab

Disable warnings

Disables warnings from the linker.

Extract common vars from libraries

Interpret uninitialized global variables defined with XCOM as definitions rather than references when processing libraries. Equivalent to the **-cd** option.

Do not create __ghs_begin symbols

Avoids outputting the COFF special symbols which are defined in the linker. Equivalent to the **-S7** option.

Relocatable program

Creates an output file which is both relocatable and executable. Equivalent to using the **-r** and **-a** command line options.

No INTEGRITY shared objs

INTEGRITY specific: do not use shared objects (libraries are statically linked into each AddressSpace).

Undefined symbols

Idenfity these symbols as undefined in the symbol table. This is generally used to force the loading of a library symbol that otherwise might not be loaded. Equivalent to the **-u *sym*** command line option.

Abs objects to link against

Use only the symbols from these objects and do not include the contents of any sections from these objects in the output file. Equivalent to the **-A *objs*** command line option.

General linker options

These options are passed through to the linker.

Generate map file

Generate a link map file showing where symbols are located. Equivalent to the **-map** command line option.

Generate cross reference file

Generate a file containing cross-reference information. Equivalent to the **-Mx** command line option.

Sort symbols by address

Sorts symbols in the link map by their address. Equivalent to the **-Mn** command line option.

Map file name

Specifies the filename of the output map file. Equivalent to using the **-map=*name*** command line option.

Cref file name

Specifies the filename of the output cross-reference information.

Map and cref directory

Specifies the directory in which the cross-reference and map files will be created.

68000 Toolchain Options > Assembler tab

Forward branch size

Sets the default size for forward branches when they've not otherwise been specified. Equivalent to **-opt: BRB, -opt:BRS, -opt:BRW, or -opt: BRL** assembler options**.**

Disable assembler warnings

Do not display warning messages on the screen. However, if a listing file is being created, the warning messages are still output to the file. By default, warning messages are output both to the screen and the listing file, if there is one. Equivalent to **-E1** assembler option.

Accept C style numerical constants

Accept C style numeric constants. Hexadecimal constants are identified by the prefix **0x**. Octal constants are identified by a leading zero. Decimal constants have no prefix. Motorola constant syntax is also accepted (**$** for hex, **@** for octal, **%** for binary). There is a conflict between this option and Motorola's normal constant analysis. Constants

beginning with zero are considered octal when this option is set but are considered decimal otherwise. Equivalent to **-opt:CNUMS**.

Reserve upper case register names

Reserves upper case register names and do not allow them to be used as variable names. Equivalent to **-reg:/UPPER**.

Reserve lower case register names

Reserves lower case register names and do not allow them to be used as variable names. Equivalent to **-reg:/LOWER**.

Reserve safe register names (such as %A5)

Reserves safe case register names and do not allow them to be used as variable names. Equivalent to **-reg:/SAFE**.

Reserve register names only for this CPU

Reserves register names that are valid only for this CPU, rather than the entire CPU family. Equivalent to **-reg:/CPU**.

Pad sections to 4-byte boundary, not 2

Pad relocatable sections to a 4 byte boundary (the default is 2). Equivalent to **-Y5.**

Use abs short mode for abs forward refs

Uses absolute short mode (16-bit) for forward references that are in the absolute format. Equivalent to **-opt:FRS**.

Interpret .L on branches to be 16-bits

Interpret the branch size code **.L** as being a 16-bit branch. Equivalent to **-opt:OLD.**

General assembler options

These options are passed through to the assembler.

Defined symbols

The format is *sym=val*. Defines a symbol '*sym*' with value '*val*'.

Generate listing file

Generates an assembly listing file. Equivalent to the **-list** command line option.

Do not expand macros

Do not expand macros in the assembly listing file. Equivalent to the **-nogen** command line option.

Do not list macro call

Do not print macro calls in the assembly listing file. Equivalent to the **-opt:NOMC** option.

Do not list macro definition

Do not print macro definitions in the assembly listing file. Equivalent to the **-opt:NOMD** option.

Do not expand control statement

Do not print expansions of control statements in the listing. Equivalent to the **-opt:NOMEX** option.

Listing file name

Specifies the filename for the listing file (optional). Equivalent to the **-list=*name*** option.

Listing file directory

Specifies the directory in which the listing file will be generated (optional).

Unix Toolchain Options > Linker tab

Relocatable program

Generates an output file which is both relocatable and executable. Equivalent to using the **-r** and **-a** command line options.

No shared objects

Prevents shared objects from resolving library references of the form **-l***name*.

Undefined symbols

Identify these symbols as undefined in the symbol table. This is generally used to force the loading of a library symbol that otherwise might not be loaded. Equivalent to the **-u *sym*** command line option.

Abs objects to link against

Use only the symbols from these objects and do not include the contents of any sections from these objects in the output file. Equivalent to the **-A *objs*** command line option.

General linker options

These options are passed through to the linker.

Map file name

Specifies the file name for the link map file (optional). Equivalent to the **-map=*name*** command line option.

Map file directory
Specifies the directory where the link map file will be generated (optional).

Unix Toolchain Options > Assembler tab

General assembler options
These options are passed through to the assembler.

Windows Toolchain Options > Linker tab

Undefined symbols
Idenfity these symbols as undefined in the symbol table. This is generally used to force the loading of a library symbol that otherwise might not be loaded. Equivalent to the **-u *sym*** command line options.

General linker options
These options are passed through to the linker.

Map file name
Specifies the filename for the map file (optional). Equivalent to the **-map**[=*name*] option.

Map file directory
Specifies the directory where the map file will be generated (optional).

Windows Toolchain Options > Assembler tab

General assembler options
These options are passed through to the assembler.

Gnu Toolchain Options > Linker tab

Relocatable program
Equivalent to linking with the **-r** and **-a** options.

No shared objects
Passes **-Bstatic** to the linker to prevent shared objects from resolving library references of the form **-l***name*. This option is available only for SunOS, Solaris2, and System V.4 Unix.

Undefined symbols
Place these symbols as an undefined symbol in the symbol table. This is generally used to force the loading of a library symbol that otherwise might not be loaded.

Abs objects to link against
The fully linked object files given are used to resolve addresses.

General linker options
These options are passed through to the linker.

Map file name
Specifies the filename for the map file (optional).

Map file directory
Specifies the directory where the map file will be generated (optional).

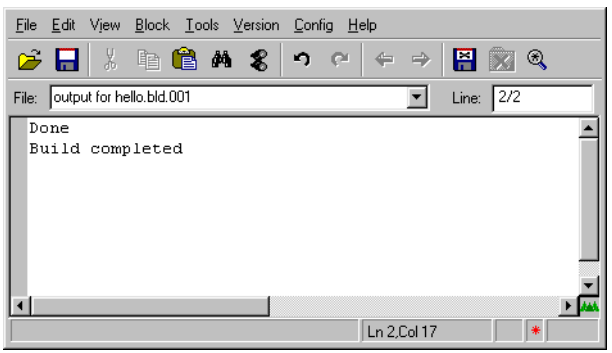
Gnu Toolchain Options > Assembler tab

General assembler options
These options are passed through to the assembler.

The Progress window

When you build a project (e.g. if you click the **Build** button to start compiling), MULTI opens a Progress window. This window displays information about the progress of the build, as well as any errors or warnings.

To control exactly what information is displayed, use the Build Panel (see “Build Panel” on page 43).



In addition to the normal editor buttons, the progress window has a **Halt** button while building. This button changes to a **Next Error** button when the build finishes.

The **Next Error** button is only present if you have either halted the build process or after the build process has completed. It allows you to quickly edit the source files which have generated errors. If any errors occurred during compilation of a source file, then pressing this button edits the file in which the error was detected in a new editor window, with the line containing the error

highlighted. Pressing the button again will do the same things, but for the next error in the build.

The **Halt** button is only present during the build. If the **Halt** button is pressed, then the build stops.

Another way to quickly edit errors is to double click the line in the progress window where the error message was reported. This will edit the source file which generated the error and highlight the line on which the error occurred. For example, double clicking an error message such as:

```
"far.c", line 6: expected: ";" got: return
```

opens a new editor window displaying the source of far.**c** with line 6 highlighted.

Chapter 4

Version control

This chapter contains:

- MULTI Version Control
- How to use MVC
- Branching and version numbers
- How to use the MVC commands
- MVC command list
- Other version control systems

MULTI Version Control

MULTI provides a proprietary version control system (MVC) for text files. By default, MULTI is configured to use MVC as its version control system. If you already use another version control system, see the “Other version control systems” on page 139 to learn how you can use your existing version control system with MULTI. If you do not have a version control system, it is easy to begin using MVC.

Caution: MVC works on text files only. If you attempt to check in any binary files, they will be deleted without warning.

MULTI keeps track of all changes to your text file in a separate log file. When you edit the file, you first “check out” your file from version control. After you finish editing your file, you then “check in” your changes. This creates a new version in the log file. The log file preserves the entire version history, allowing you to restore any previous version to compare or revert back to another file.

If you are using the MULTI editor to edit a file under version control, the file is automatically checked out when you make changes, assuming you have Automatic Checkout enabled. The file is automatically checked in when you close it. During the check in process, a dialog appears, requesting comments. These comments are saved in the log file along with the new version.

To prevent changes from being made to files without the version control system’s knowledge, all files in version control are read-only. Files become writable when they are checked out and return to read-only status when they are checked in. Files become writable only to the user who checked out the files. This prevents multiple users from editing the same file at the same time.

When a file is in version control, several directories are created in the same directory:

mvc.log
: This directory contains log files. For example, if the file /**a/john/fly.c** is placed under version control, then the log file is **/a/john/mvc.log/fly.c**.

  Note: These log files must not be modified by hand, as all version history may be lost if the file becomes corrupted.

mvc.log/mvc.lok
: This directory contains the lock files showing who has checked out the file. MVC only allows one user to check out a file at a time. The lock file for /**a/john/fly.c** is **/a/john/mvc.log/mvc.lok/fly.c**.

mvc.log/mvc.sem
: This directory contains temporary files used as semaphores to prevent two different MULTI sessions from writing to the same log file at the same time.

How to use MVC

There are three ways to access MULTI version control:

1. Use the Editor on a text file under version control. Files are checked in and out automatically. Although MVC is enabled by default, only your **.bld** files are automatically put under version control. All other files must be placed under version control manually. To place files under version control:
    a. Select Editor > Version > Place Under VC.
    b. Enter the **create** command on the command line.
2. Choose menu items in the Builder or Editor Version menus.
3. Enter a full MVC command line. You can enter a full MVC command in two different places:
    a. The first is from a UNIX shell; there is an MVC executable which comes with MULTI.
    b. Choose Version > Other VC Command... in the Builder.

Example

```
% mvc co foo.c
```
(assumes mvc is in your PATH)

Branching and version numbers

Version numbers are created when new files are added to a branch or version. This allows you to revert to previous versions if necessary. Version numbers have the form:

major_version.*minor_version*

As you create new versions, the minor version number increments by one for each new version. You can set the version number for a new version (for instance if you want to increment the major version number) by using the **-v** version option of the various check in commands. This is done from an MVC command line. For example, **mvc ci foo.c -v 3.2** checks in **foo.c** with version

number 3.2. Version numbers are never allowed to decrease. 1.4 -> 1.5 and 1.5 -> 2.1 are allowed, but 2.5 -> 2.4 and 2.5 -> 1.6 are not allowed.

You can create a branch in the version for a file, by adding two more dots (**..**) to the version number. For example, a version sequence might be: **1.1**, **1.2**, **1.3**, **1.4**. If you create a version with the number **1.3.1.1**, that version is a branch off the main sequence. The version sequence for that branch would be **1.1**, **1.2**, **1.3**, **1.3.1.1**. The next version for that branch after **1.3.1.1** would be **1.3.1.2**.

To create a branch, you use the **-v** version option of the various checking commands. The version is of the form of a branch version (with 4 numbers). This needs to be done from the MVC command line. For example, **mvc ci foo.c -v 1.5.1.1** creates a branch off of version 1.5. When you work on a branch, you always use the **-v** version option. If you omit the **-v** version option, MVC assumes that you are working on the main version sequence. This creates a version on the main sequence instead of on the branch. For example:

mvc co foo.c -v 1.5.1.1

(edit **foo.c**)

mvc ci foo.c -v 1.5.1.2

NOT **mvc ci foo.c**, which creates a version on the main branch (**1.8**, for instance).

How to use the MVC commands

Some of the MVC commands are also listed in the Editor > Version and Builder > Version menus. To type these commands, see “How to use MVC” on page 131.

For any of the commands that check in a file, you can enter the **-c** option to force MVC to ask for comments.

In all of the following commands, *version* refers to the version number you want the command to manipulate. If no version number is specified, then the current version is assumed. For a check out, the current version is the latest version on the main version sequence. For a check in, the current version is the same but with the minor version number incremented by one.

Some of the commands use a date (**-d** *date*) instead of a version number to refer to a given version. The date needs to be in the form:

MMDDYYhhmmss

where:

MM	month
DD	day
YY	year
hh	hour
mm	minutes
ss	seconds

The individual components of the date are separated by non-digit characters (except for white space). For example, you can specify the date as **082597120000** or **08.25.97.12.00.00**. Do not put spaces in the date. If any part of the date is omitted, the maximum value for that part is used. For example, **082597** implies **082597235959**. Two digit years between 50 and 99 are assumed to describe years from 1950-1999, while years from 00 to 49 are used to refer to years from 2000-2049.

You can use four-digit years by using the **-D** *date* option which takes a date in the form:

YYYYMMDDhhmmss

All of the following commands use a *filename*. Multiple filenames are separated by spaces. You can specify these files using the command-line option **-l** *list_file* instead of *filename*. The list file is a normal text file that starts with the keyword **mvc-list**, followed by the number of files, then the filenames. Everything in the list file needs to be separated by spaces, tabs, and/or newlines. For example, you have the following list file named fly:

```
mvc-list 3
art.c
trip.c
hat.c
```

With the above list file, you can specify a command to work on all three of the listed files, **art.c**, **trip.c**, and **hat.c**. This means the following two commands are identical:

```
ci art.c trip.c hat.c
ci -l fly
```

With both of these commands, the three files are checked in. The first line lists them explicitly; in the second line, a list file is specified which contains the three files.

To use directories other than the current one, specify the **-L** *logdir* and **-S** *sourcedir* options. *logdir* is the directory that contains the **mvc.log** directory to use for the log files. *sourcedir* is the directory containing the source files. For example, if the source files are in **/usr/john** and the log files are in **/usr/george/mvc.log** then the command:

```
ci fly.c bat.c -S /usr/john -L /usr/george
```

takes the files **/usr/john/fly.c** and **/usr/john/bat.c** and checks them into the log files **/usr/george/mvc.log/fly.c** and **/usr/george/mvc.log/bat.c**.

MVC command list

Alias

alias *filenames* **-v** *alias* [**-V** *version*]

Allows you to refer to a given version number in the log file for the files specified by *filenames* by another name, *alias*. This alias is used with other MVC commands to refer to that version. If you do not specify **-V** *version*, the current version is assumed. Valid aliases must satisfy the following conditions:

- they must be one word;
- they must begin with a non-digit;
- two aliases with the same name cannot be defined for the same file.

For example, if version 5.3 is a working version, enter:

```
alias fly.c -v goodone -V 5.3
```

where the word **goodone** becomes the alias for version 5.3. Later, you can use **goodone** to specify the version number in any command which takes a version number as an argument. For example:

```
get fly.c -v goodone
```

This command is extremely useful for making a whole set of source files. For instance, if you are ready to release a product and want to mark all the current sources as *release*, make sure all source files are checked in and then do the following:

```
alias [list of source files] -v release
```

Then you can continue working on the product, making changes and creating new versions. If you ever need to return to the sources for the release version, you can use the following:

```
get [list of source files] -v release
```

Another way to accomplish the same goal is to specify the date of the release:

```
get [list of files] -d date
```

Remembering an *alias* is easier than remembering a date. If you find that you need to change the alias (for example, if a new release is going to replace the old one), you can use the **unalias** command to remove the old alias, then use the **alias** command again to create the new alias.

Copy file

copyfile [*list of files*] *new_directory*

Copies each file and its accompanying log file to the *new_directory.*

Create log

create *filenames* [**-v** *version*]

Creates a log file for each file specified by *filenames*. The log file is placed in the **mvc.log**, which is created if it does not currently exist. The log file's name is **mvc.log/filename**.

You can specify the starting version number with **-v** *version.* The version numbers start at 1.1 by default.

Check in changes

All these commands check in changes made to the file into the log file. For this command to work, the file needs to be previously checked out.

delta *filenames* [**-v** *version* | **-d** *date*]
: Changes are checked in and the file is deleted.

delget *filenames* [**-v** *version* | **-d** *date*]
: Changes are checked in and the file becomes read-only.

ci *filenames* [**-v** *version* | **-d** *date*]
: Same as **delget**.

deledit *filenames* [**-v** *version* | **-d** *date*]
Changes are checked in, but the file remains checked out and is editable. In effect, this is a check in followed by a check out.

cio *filenames* [**-v** *version* | **-d** *date*]
Same as **deledit**.

Delete file

mvc deletefile *filenames*

Deletes the *filenames* and its log files.

Diff Files

diff *filename* [**-v** *version1* **-V** *version2*]

Finds the differences between different versions of the specified file. To specify the versions, enter one of the following:

diff *filename*
Determines the latest changes to *filename*. MVC finds differences between the source file version retrieved from the log file and the last version in the log file. If the last version in the log file is the same as the source file, it uses the next to the last version instead. If the source file does not exist, MVC compares the last version in the log file to the next to the last version in the log file.

diff *filename* -v *version*
Compares the latest version of *filename* against the specified version. MVC considers the source file the latest version. If the source file does not exist, the latest version in the log file is used.

diff *filename* -v *version1* -V *version2*
Compares *version1* of *filename* to *version2 of filename.*

Display version

disp *filenames* [**-v** *version* | **-d** *date*]

Displays the files specified by *filenames* and their versions. The version number in which each line was originally created is prepended to the line.

Check out and edit

edit *filenames* [**-v** *version* | **-d** *date*]
co *filenames* [**-v** *version* | **-d** *date*]

Check out the specified files and retrieve editable copies for your use. These commands are identical.

Find changed version

fc *filenames* [**-v** *version* | **-d** *date*] **-s** *startline* [**-e** *endline*]

Finds the most recent version of the files specified by *filenames* which changes a particular piece of text. *startline* specifies the starting line number of the change. *endline* specifies the ending line number of the change. If no ending line is specified, then only one line (*startline*) is used. See also "ShowLastEdit" on page 210. The current file must also be checked in. Otherwise, the results of this operation are undefined.

Read (only) version

get *filenames* [**-v** *version* | **-d** *date*]

Retrieves a read-only copy of the specified file. This does not check out the file and is not affected if someone else checks out the file. This command does not work if you are currently editing a writable copy of the file.

Move file

movefile *filenames new_directory*

Moves each file and its log file to the *new_directory.*

Remove from version control

unmvc *filenames*

Removes *filenames* from version control.

Package files

package [**-f** *packfile*] *filenames*

Archives, or "packs", the listed files and their corresponding log and lock file if they exist. The result is placed in the file **logs.pak**, unless you specify *packfile*. The package file is in UNIX tar format.

Unpackage files

unpackage [**-f** *packfile*] [**-L** *directory*] [*filenames*]

Unpacks the files in the specified *packfile*, or from **logs.pak** if a *packfile* is not specified. If you specify *directory*, all the files unpack into that directory. If you specify *filenames*, then only the specified files are unpacked.

Delete version

remver *filenames* [**-v** *version* | **-d** *date*]

Deletes the specified version.

Important: Do not delete any versions that start branches or you will not be able to trace the branched copies back to the originator. Once a version is deleted it can never be recovered, so use this command with care.

Show log

show *filenames* [**-F**]

Displays a table of contents of the log files for the files specified by *filenames*. Each version in the log file is displayed with its date, username, and comment. By default, only the first line of the comment is displayed. To display the full comment, specify the **-F** option.

Unalias

unalias *filenames* **-v** *alias*

Removes an alias (*alias*) previously defined with the **alias** command.

Check in, lose changes

uncheck *filenames*

unedit *filenames*

Checks in a file without checking in the changes, so that any changes made to the file are lost. This command applies only to files which have previously been checked out. A read-only copy of the latest version of the file is then retrieved. This command is useful if you check out a file and later decide you don't need to make any changes to it.

Unlock file

unlock *filenames*

Forcibly checks in a file, even if the file was checked out by another user. Any changes made are checked in. This command is useful when someone

accidentally leaves a file checked out and cannot be contacted to check the file back in. If the file was checked out by another user, a mail message is sent to that user.

Who checked out a file

who *filenames*

Displays the user who has each file specified by *filenames* checked out, as well as the time when each file was checked out.

Other version control systems

MULTI supports several version control systems to keep track of your source code changes:

- MVC
- RCS
- ClearCase

MULTI Version Control (MVC) is provided with MULTI for those who do not have another version control system. RCS and ClearCase are available from third-party vendors. If you use a version control system not specified above, you will not experience the benefit of MULTI's integrated version control features.

How to use other version control systems with MULTI

MULTI provides the Version menu in both the Builder and Editor Windows where you can perform version control operations such as checking in files, checking out files, and showing histories. For more information on Version menu options, see "Version menu" on page 39 in the Builder or "Version menu" on page 170 in the Editor.

When you begin to edit a file in the Editor, MULTI automatically checks out the file for you if Automatic Checkout is enabled. See "Check or uncheck 'Automatic checkout'." on page 140. When you close a file you have checked out during an editing session, MULTI asks you whether to check in the file or leave it checked out.

When you open a project in the Builder, any checked out files in the project contain the user's name listed next to the file.

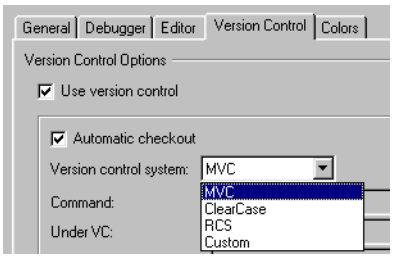
When you debug a program with MULTI, it attempts to find the version of the source files which was current when the program was built. Files from version control may be used if the source files have changed since the program was built.

If you are using ClearCase as your version control system, you need to set the view you want to use before you start MULTI. You cannot change it once you have started MULTI. However, you can see what your current view is by choosing Version > Show View in the Editor.

To enable other version control systems with MULTI

For MVC, RCS, or ClearCase Users

1. Choose Config > Options... > Version Control tab.
2. Check the 'Use version control' checkbox. (See also "Options..." on page 232.)



3. Choose the appropriate version control system from the Version control system drop-down list.
4. Check or uncheck 'Automatic checkout'.

If you enable 'Automatic checkout', MULTI automatically checks out a file for you when you start editing. If this box is not checked, choose Version > Check Out and MULTI allows you to edit a file. This setting only affects new files that you open; files already open when you change this setting are not affected. To change this attribute on a per-file basis, choose Version > Auto Checkout.

Note: If you use RCS, then **ci**, **co**, **rlog**, and **rcsdiff** must exist in your path. If you use ClearCase, then **cleartool** must exist in your path.

Tip: Save your configuration and restart MULTI whenever you change either the "Version control system" setting or the "Use version control" setting. Switching version control systems while you have files open can lead to unpredictable results.

For other version control systems

1. Choose Config > Options... in the Builder, Editor, or Debugger.
2. Uncheck the Use version control checkbox under the Version Control tab. (See “Options...” on page 232 for more information.)

Note: MULTI provides special support and added benefits for ClearCase, RCS and MVC users. If you use another version control system, you may be able to create custom menus that control your version control system. This can give you some (but not all) of the benefits of using a more supported version control system. For more information, see Appendix A, “Third party tools”.

Tip: Save your configuration and restart MULTI whenever you change the “Version control system” setting or the “Use version control” setting. Switching version control systems while you have files open can lead to unpredictable results.

Chapter 5

Using the Editor

This chapter contains:

- Starting the Editor
- Opening files
- Saving files
- Editing
- Working with your code
- Searching
- Merging files
- Comparing files
- Using version control from the Editor
- Configuring the Editor

Starting the Editor

You can start the Editor from other MULTI tools or as a standalone editing program.

To start the Editor from the Builder window

As you use the Builder to navigate through your projects, you can open a source file the following ways:

- Double-click the filename in the Source pane.
- Select one or more files in the Source pane, then choose Project > Edit Selected Files to edit the selected files.
- Choose File > Open File in Editor to be prompted for the name of an arbitrary file to edit.

  If the file already exists, then the Editor will open the existing file; otherwise, the Editor will open on a new (blank) file.

To start the Editor from the Progress window

When you use the Builder to build a project, a Progress window appears with information about the build, including any build errors. When you double-click an error in the Progress window, the Editor opens the source file, placing the cursor on the line with the error.

To start the Editor from the Debugger

When you start the Editor from the Debugger, MULTI creates temporary copies of source files so you can continue to debug your program while looking at the original source code. Changes that you make in the Editor affect the actual file, not the temporary file that the Debugger is using to show the original code. When you exit the Debugger, the temporary files are deleted.

To open the Editor on the Debugger's current source file, click the Edit button in the Debugger.

You can also start the Editor by entering commands in the Debugger Command Window. The following table summarizes which command to use to open a certain file:

Debugger commands that start the Editor		
To open	**Enter**	**Examples and Comments**
The current source file	**edit**	n/a
A file by name	**edit** *filename*	n/a
A file by selecting it in a dialog box	**editfile**	n/a
A file that contains a certain procedure	**edit** *procedure*	If a procedure name is followed by a wildcard pattern, then a window appears with a list of procedures from which you select a procedure to edit. For example, the command **edit f*** opens a window that lists all procedures beginning with the letter **f**.
A file whose procedure has a certain breakpoint	**edit** *number***b**	For example, to edit the procedure containing breakpoint number three, use the command **edit 3b**. You can obtain breakpoint numbers with the **B** command.
A file whose procedure is at a certain stack depth	**edit** *number_*	For example, to edit the procedure at stack depth three, use the command **edit 3_**. You can obtain stack depths with the **calls** command.

To start the Editor as a standalone program

The Editor is an executable program, **me**, that is located in the Green Hills directory (by default, /usr/green). To run the Editor as a standalone program, you can:

- Enter the following at a command line prompt:

  me *+linenumber filename*

Opening files

As you are working in the Editor, you can open a new file in the current Editor or in a separate Editor window.

To open a file in the current Editor window

You can open multiple files in the same Editor window. When you open an additional file, the Editor places the newly opened file on top of its stack of open files. You can then navigate through the open files using View > Next File and View > Previous File, or the toolbar buttons.

1. Choose File > Open
   – or –
   Click the Open button( )
2. In the Edit File file chooser, select the file you want to open.

Shortcut: To quickly open a file, type the filename in the File field, and press Enter.

If the file is not located in the current directory, you must include the path.

To open a file in a new Editor window

Some people prefer to work with multiple Editor windows, each containing a single file, rather than opening multiple files in the same Editor window. If you are working in the Editor and want to open a different file in a separate Editor window:

1. Choose File > New Editor.
2. In the Edit File file chooser, select the file you want to open.

Shortcut: To quickly open an existing file in a different Editor window, type the filename in the File: field, and press Shift+Enter. If the file is not located in the current directory, you must include the path.

To create a new file

If you want to create a new file, follow the first step to opening an existing file. When the Edit File file chooser appears, enter a new filename and click Open. The Editor creates and opens the new file.

Shortcut: To quickly create a new file in the current directory, type the filename in the File field, and press Enter. If you want the new file to open in a new Editor window, press Shift+Enter.

Navigating between open files

When you open multiple files in the same Editor window, only one file is visible at any given time. The rest of the open files are stacked below the current file in the order in which they were opened.

To view the previous file

To view the file that you were looking at just prior to the current file, choose View > Previous File, or click the Previous File button ().

To view the next file

To view the next file in the Editor's stack of open files, choose View > Next File, or click the Next File button ().

If you are viewing the most recently opened file, going to the next file allows you to view the first file opened.

Navigating between files in different Editor windows

If you like to work with multiple Editor windows, each containing a single file (rather than opening the files in the same Editor window), you might want to customize the Editor to make the **NextWindow** command (see "NextWindow" on page 215) easily accessible. You can use this command to cycle through all of the Editors that are currently open on your computer. To learn how to add commands to the Editor through menus, buttons, mouse clicks, and keystrokes, see Chapter 9, "Configuring and customizing MULTI".

Saving files

There are several ways to save changes made to the file or files being edited.

To save changes to the file currently being viewed

To save changes to the file currently visible in the editor, do one of the following:

- click the Save button ()
- choose File > Save

To save the file currently being viewed under a new name

To save the file currently visible in the editor with a different name, choose File > Save As.

To save all files currently open in the editor

To save all the files currently open in the editor, choose File > Save All.

Editing

To perform common editing operations

You can right-click in the editor window to open a menu of common editing operations. This menu allows easy access to cutting, copying, and pasting the current selection, undoing the last editing operation, or navigating to the tag the cursor is currently over. All these functions are documented separately in this chapter.

To reverse changes made to a file

You can reverse any edits you have made to a file since you opened it.

To reverse the last edit, do one of the following:

- Click Undo ().
- Right-click and choose Undo from the pop-up menu.
- Choose Edit > Undo

Keep clicking Undo to reverse more edits, until the file is in the same state as when you opened it.

To restore changes that you reversed

To restore any changes you have reversed using Undo, do one of the following:

- click the Redo button ()
- choose Edit > Redo

Each time you choose Edit > Redo, the effects of one Undo are reversed, beginning with the most recent.

To reverse all changes made to a file since the last save

To return to the last saved version of a file, choose File > Revert to Saved.

To insert a character blocked by a custom keybinding

If you customize the Editor to run a command based on a single keystroke, you cannot directly insert the literal character of that keystroke into a file. For example, if you customize the Editor so that every time you press **d** the cursor moves down one line, then you will not be able to type the literal letter **d** in a file. In this scenario, you would have to complete the following steps to enter the literal character **d** in your file.

1. Press Ctrl+\
2. Press the key for the character that you want to enter into the file.

To repeat the last change you made to a file

1. Place the cursor where you want to repeat the last edit.
2. Choose Edit > Repeat Last Edit.

You can repeat only certain types of edits. For example, if you just selected text and then replaced it with new text, then repeating the last edit will delete a similar selection contiguous to the cursor and insert the new text. Suppose you file contains the text:

```
The albatross said it was 8:09, and everyone cheered.
```

If you select the word **everyone** and type **rainbow**, your text becomes:

```
The albatross said it was 8:09, and rainbow cheered.
```

If you move the cursor to the beginning of the word **albatross** (you do not have to highlight the word) and choose Edit > Repeat Last Edit, the text becomes:

```
The rainbow said it was 8:09, and rainbow cheered.
```

To copy a column of text

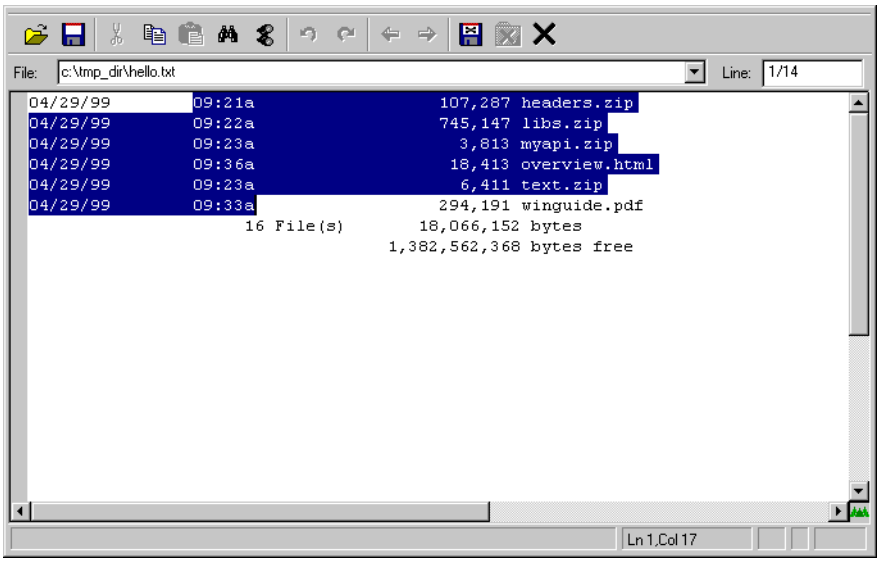
You can copy a column of data from a file, excluding data on either side of the column. For example, suppose you have a tab delimited file that lists date, time, file size, and filename. You can copy just the column that list the times without affecting any of the other data.

1. On the first line that contains data you want to copy, start the selection at the first character you want copy.
2. Extend the selection to include all of the data in the column. The last characters selected should be the last character of the column.

3. Choose Block > Rect Copy.

Example

You highlight the following selection, then choose Block > Rect Copy.



When you paste the contents of the clipboard, the following text is inserted in your file:

```
09:21a
09:22a
09:23a
09:36a
09:23a
09:33a
```

To cut a column of text

To cut a column of text out of a file, highlight the column and choose Block > Rect Cut. For more details on how to highlight a column of text, see “To copy a column of text” on page 149.

To paste a column of text

After you have cut or copied a column of text from a file, if you use the regular paste (Edit > Paste), the contents of the clipboard are pasted in the file with line breaks.

To paste the column without inserting line breaks, choose Block > Rect Paste.

Working with your code

To configure the Editor for your programming language

When you open a source file that has a language-specific extension, the Editor automatically configures itself to work with that programming language. For example, if you open a file **foo.c**, the Editor uses **/* */** whenever you insert a comment block. The Editor also displays elements in your code in different colors based on the specified language.

If you need to manually specify your programming language because the Editor does not recognize a file extension:

1. Open the source file.
2. Choose View > Language, then select the programming language used in the source file.

Using comments

To insert a comment

The Editor inserts the proper syntax for comments based on the programming language you are using. If the Editor is not using the correct syntax, choose View > Language to make sure it is set to the correct language. Block comment operations are not supported in FORTRAN or Green Hills Script.

First, do one of the following:

- To comment out existing code, highlight the text. If there is no highlighted selection, the entire line the cursor is currently on will be commented out by default.
- To insert a new comment, place the cursor on a blank line where you want the comment to start.

Then, do one of the following:

- Right-click the highlighted code, and choose Comment from the pop-up menu.
- Choose Block > Comment.

If you want to uncomment a comment block, highlight the block and choose Block > UnComment or right-click the selected block and choose Uncomment from the pop-up menu.

To keep comments flush-left

If you want comments in your code to stay flush-left even when you auto-indent your code:

1. Choose Config > Options....
2. On the Editor tab, select Comments Stick Flush Left.
3. Enter your comments next to the left margin.

Note: If you insert # as the first character of a comment, then the comment will move to the left margin regardless of the position where you started the comment. For example, if you are coding in C and enter **/*#**, then the comment automatically moves to the left margin.

Indenting your code

As you write code, you can insert an indent manually, or you can let the Editor indent your code based on common coding standards.

To set the size of indents code

You can change the size of indents that you manually insert, or that are automatically inserted by the Editor.

1. Choose Config > Options....
2. On the Editor tab, change the Indent size field to specify the size of indents.

To manually insert or remove an indent

1. Place the cursor on the line of code you want to indent or unindent.
2. To insert an indent, choose Block > Indent.

   To remove an indent, choose Block > Unindent.

To let the Editor indent your code

The Editor has an auto-indent feature that indents your code according to common coding standards.

1. If you want to auto-indent a single line of code, move the cursor to that line. If you want to auto-indent multiple lines of code, highlight those lines.

2. Choose Block > Auto-Indent, or press Ctrl+2, or press Ctrl+; (semi-colon). Pressing Tab also automatically indents everything to the right of the cursor.
3. Select the block you wish to indent, right-click it, and choose Auto Indent from the pop-up menu.

Influencing how the Editor auto-indents your code

You can change how far the Editor auto-indents the lines of a lexical block of code. At the start of the lexical block, indent the code how you want the entire block to look. Then, highlight the rest of the block and start the auto-indent. This prevents the Editor from breaking the conventions of a pre-existing block.

How indenting multiple lines affects your comments

If you are auto-indenting multiple lines of code and do not want to indent the comments within those lines:

1. Choose Config > Options....
2. On the Editor tab, deselect Indent Comments when Indenting Multiple Lines.

Characters that auto-indent your code

By default, the Editor automatically makes indenting adjustments when you enter the following characters:

#	:
*	{
;	}

To disable characters from auto-indenting your code and comments

1. Choose Config > Options....
2. On the Editor tab, deselect Implicit Auto Indent.

To disable characters from auto-indenting your comments only

You might want special characters to auto-indent your code, but to disable them if you use them within a comment.

1. Choose Config > Options....
2. On the Editor tab, deselect Implicit Auto Indent In Comments.

Indenting the line following a left parenthesis '('

You can configure how the Editor indents the line of code that follows a left parenthesis '('.

1. Choose Config > Options....
2. Go to the Editor tab.
3. In the C Paren Indent Mode, select one of the following according to your preferred coding standard:
   - If you want the Editor to indent by two levels the line of code that follows a left parenthesis '(', select "Indent in two". Your code will look like this:

     ```
     int main (
                         int argc
     ```

   - If you want to indent the line of code that follows a left parenthesis '(' so that it lines up with the parenthesis, select "Even with parentheses". Your code will look like this:

     ```
     int main (
              int argc
     ```

To alter the case of the currently selected code

Choose Block > UpperCase to transmute the case of all alphabetic characters in the current selection to uppercase, or Block > LowerCase for lowercase. This operation is not supported in FORTRAN, Pascal, and Green Hills Script.

To highlight the boundaries of the current block of code

To quickly identify the start and end of the current block of code, choose View > Match.

The Editor searches backward from the cursor and finds the first enclosing instance of a left parenthesis '(', left curly brace '{' , or a left bracket '[' , then searches forward from the cursor to find the matching ending mark and selects the code in between.

Using tags in your files

If you use the utility **ctags** to create a tag file, the Editor uses the information to open files and move the cursor based on the name of a function.

To navigate to a function

1. Choose Edit > Goto...
2. In the GoTo dialog box, select the Function radio button. If the Function radio button is not available, the Editor could not find a tag file.
3. Type the name of the function, and click Go. If the function is in the current file, the Editor moves the cursor to the beginning of the function. If the function is in a different file, the Editor opens that file and moves the cursor to the beginning of the function.

— or —

1. Right-click the function name and choose Jump to Function from the pop-up menu.

To manually load a tag file in an Editor session

When the Editor starts, it looks for a file called **tags**. If the tag file has a different name or is in an unexpected location, the Editor may not find it. To manually load a tag file into an Editor session:

1. Choose Tools > Append TagFile.
2. Enter the path and filename of the tag file, and click OK.

To remove a tag file from an Editor session

If you want to unload a tag file from an Editor session, choose Tools > Reset Tags. Once you have unloaded a tag file, the Editor no longer uses that file to navigate to functions.

Searching

To start a full search, do one of the following:

- Click Search ( ).
- Choose Edit > Find...

For a general description of the fields available to tailor a search, see “Search dialog box” on page 178.

To make a “quick” incremental search

If you do not want to spend the time opening the Search dialog box and do not need to replace what you find, you can perform a quick search.

1. If you want to search forward in the file, press Ctrl+f.
   If you want to search backward in the file, press Ctrl+b.
   The left corner of the status bar changes to **Srch:**.
2. Enter the character pattern you are looking for. As you enter the characters, the Editor highlights the first occurrence of that character pattern.
3. If the current file contains more than one occurrence of the character pattern you entered, continue to press Ctrl+f to view the next match or Ctrl+b to view the previous match until you find what you are looking for.

Quick search tips

- If you want quick searches to be case-sensitive, choose Config > Options..., go to the General tab, and select Match Exact Case in Searches. Be aware that this setting affects quick searches in all MULTI tools, not just the Editor.
- If you previously used the Search dialog box to perform a full search, you can perform a quick search using the same advanced criteria. To perform a quick search, press Ctrl+f, then press Ctrl+f again without entering any text. The Editor searches for the first instance that matches the criteria previously defined in the Search dialog box.

To search using wildcards

1. Choose Edit > Find..., or click Search ( ).
2. In the Search dialog box, select Wildcard.
3. Enter the character pattern you are looking for. The following characters act as wildcards:

Wildcard	Behavior	Example
?	Matches any single character, except newline	**c?t** finds **cat** and **cot**, but not **coat**
*	Matches any number of characters, except newlines	**c*t** finds **cat**, **cot**, and **coat**

Merging files

You can use the Editor to merge two or three files into a single file. The two or three files can be different versions of the same file or be different files.

If you are using a version control system and want to use a different version of the file from disk, then enter the name of the file which you want to merge in

the Filename field, and the version of that file you wish to merge in the Version text field.

To merge two files into a single file

1. Choose Tools > Merge Files...
   An EditMerge dialog box appears. In the File1 field, enter the first version of the file. If you specify a file which you are currently editing and you do not enter a version number, File1 will be the copy of the file with your current (unsaved) edits. This is useful if someone else has edited the file at the same time you were working on it.
2. In the File2 field, enter the second version of the file. If you specify a file which you are currently editing and you do not enter a version number, File1 will be the last saved copy of the file.
3. Deselect Automatic.
4. Click Merge.

   A window appears for each file you specified, as well as an extra window for the results of the merge. To identify which file is in a window, look at the title bar at the top of the window.
5. Use the Merge window to select what gets placed in the merged file.

   The Editor pauses at each point where the two files are different, and highlights the text that differs. Using the Control Panel, you can select which one of the highlighted sections, or both, to copy into the results window. You can also manually cut and paste text into the results window. Listed below are the features of the results window:

Skip
: The Editor finds the next difference.

Help
: Opens **help** on the control panel.

Cancel
: Aborts the merge, closing all merger windows.

File1
: Copies the selected text from File1.

File2
: Copies the selected text from File2.

Both

The first time you press this button, a dialog box appears asking you how you want to merge the two selections: in what order, with change bars around them, with comments in front, between, or after them, and so forth. The next time you press this button, the last values entered are used.

Change Bars...

Changes the way the **Both** button works, and opens the same dialog box as the first time you press the **Both** button.

When merging is complete, the windows on the original files are removed and a dialog box allows you to save the results window. If you save it as the same name as one of the original files, then any windows still looking at that file are replaced with the merged results. After saving, the results window is removed.

To merge three files into a single file

When merging three files, one is considered the base file that the other two are derived from. With this assumption, the Editor is usually able to merge without asking you. The Editor does this using the following rules:

- If a difference exits between the two source files, and one is the same as the base file, then the Editor uses the one that is different from the base file.
- If both source files differ from the base file, but are the same as each other, then the Editor uses the new text from either source file.
- If all three files are different, then a conflicting change was made and the Editor has to ask which change to use. In this case, it is likely that you will have to merge the change manually.

1. Choose Tools > Merge Files...
2. In File1, enter the first variation of the base file. If you want to use a version from version control, enter the version number in the adjacent Version field.
3. In File2, enter the second variation of the base file. If you want to use a version from version control, enter the version number in the adjacent Version field.
4. In Base, enter the version of the file from which File1 and File2 files are derived. If you want to use a version from version control, enter the version number in the adjacent Version field.
5. If you want to manually control every merge change, deselect Automatic. If Automatic is selected, the Editor asks you to control a change only if it encounters a conflict it cannot resolve.

6. Click Merge.
7. Use the Merge window to control merges. If you deselected Automatic, the Merge window appears for every change. If you selected Automatic, the Merge window appears only when the Editor encounters a conflict it cannot resolve. Listed below are the features of the merge window:

Skip
The Editor finds the next difference.

Help
Opens **help** on the control panel.

Cancel
Aborts the merge, closing all merger windows.

File1
Copies the selected text from File1.

File2
Copies the selected text from File2.

Base
Copies the selected text from the base file.

All
Copies the selected text from all three files.

The first time you press this button, a dialog box appears asking you how you want to merge the three selections: in what order, with change bars around them, with comments in front, between, or after them, and so forth. The next time you press this button, the last values entered are used.

1 & 2
The first time you press this, or either of the next two, a button dialog box appears asking you how to merge the two selections from File1 and File2: in what order, with change bars around them, with comments in front, between, or after them, and so forth. The next time you press this button, it uses the last values entered.

1 & Base
This is identical to **1 & 2** except it merges the selections from File1 and the base file.

2 & Base
This is identical to **1 & 2** except it merges the selections from File2 and the base file.

ChangeBars...

Changes the way **1 & 2**, **1 & Base**, and **2 & Base** buttons work, and opens the same dialog box as the first time you press any of those buttons.

Automatic

Choose if you want the Editor to try to make merge changes without prompting you. Deselect if you want to manually control every merge change.

Comparing files

To compare two files, do the following:

1. Choose Tools > Diff Files...
2. In File1, specify the version you want to compare to File2. If you do not specify a version, the Editor assumes you mean the current version.
3. In File2, specify the version you want to compare to File1. To decrement the version number, click Previous Version.
4. Click Diff.

   The specified versions appear in a separate windows.
5. Click Previous or Next to navigate among the differences between the two versions. The Editor highlights the differences.

Using version control from the Editor

The Editor is fully aware of several version control systems. Many version control operations can be performed without leaving the Editor when using a supported version control system.

To configure MULTI to work with your version control system

See Chapter 4, “Version control” for more information.

To automatically check out files when they are modified

The Editor will prevent you from making changes to files that are not checked out, because they are read-only. To configure the Editor to automatically check out files when you modify them, choose Version > Auto Checkout or choose

Config > Options... and check or un-check Automatic Checkout on the Version Control tab.

To check out a file manually

Choose Version > Check Out to check a file out of version control.

To save your changes and check in a file

There are several ways to check in files. To save a current file and manually check it in, choose Version > Check In. To manually check in all open files that are checked out, choose Version > Check In All. Finally, when you close a file or the editor, you will be asked whether to check in any modified files which you have checked out during the current editor session.

To check in a file and revert to the previous version

Choose Version > Discard Changes to check a file back in without making changes.

To put a new file under version control

Choose Version > Place Under VC.

To view the version history of a file

Choose Version > Show History to display the list of file versions and the comments made when these versions were checked in.

To show the last change to a portion of a file

This feature is only supported when using MVC as the version control system. Select the portion of the file you are interested in and choose Version > Show Last Edit. Two additional editor windows will appear displaying the version of the file where the last edit to the selected area was made and the version immediately prior to that version. A third window will appear allowing you to navigate between changes made at the time of the last edit to the selected area.

Reverting to a previous version of a file

There are three ways to specify the version of a file you wish to revert to:

- If you wish to select the version to revert to from a list of all versions, choose Version > Revert to History.
- If you wish to revert to the current version as of a particular date, choose Version > Revert to Date.
- If you know the version number you wish to revert to, choose Version > Revert to Version.

Configuring the Editor

To help you work more efficiently, you can:

- Change how the Editor looks and behaves. For example, you can change whether you can drag and drop text within a file. To access settings that change how the Editor looks, behaves, and handles your code, choose Config > Options..., and go to the Editor tab.
- Customize the Editor to perform actions in new ways. You can replicate all of the actions that are available in the standard Editor window by assigning the appropriate commands to a new menu, keystroke combination, mouse click combination, or button. For example, if you do not like to use the mouse when you are editing, you can assign commands to keystrokes.

Perhaps the easiest way to find out what command you want is to look in Chapter 5, “Using the Editor”, which lists the equivalent command for each GUI component. You can also look at the complete list of commands in Chapter 7, “Editor commands”.

For information about how you assign commands to menus, keybindings, mouse bindings, and buttons, see Chapter 9, “Configuring and customizing MULTI”.

Once you learn the basic method of customizing menus, keybindings, mouse bindings, and buttons in the Editor, you can easily customize the Editor to improve your efficiency.

Chapter 6

The Editor GUI

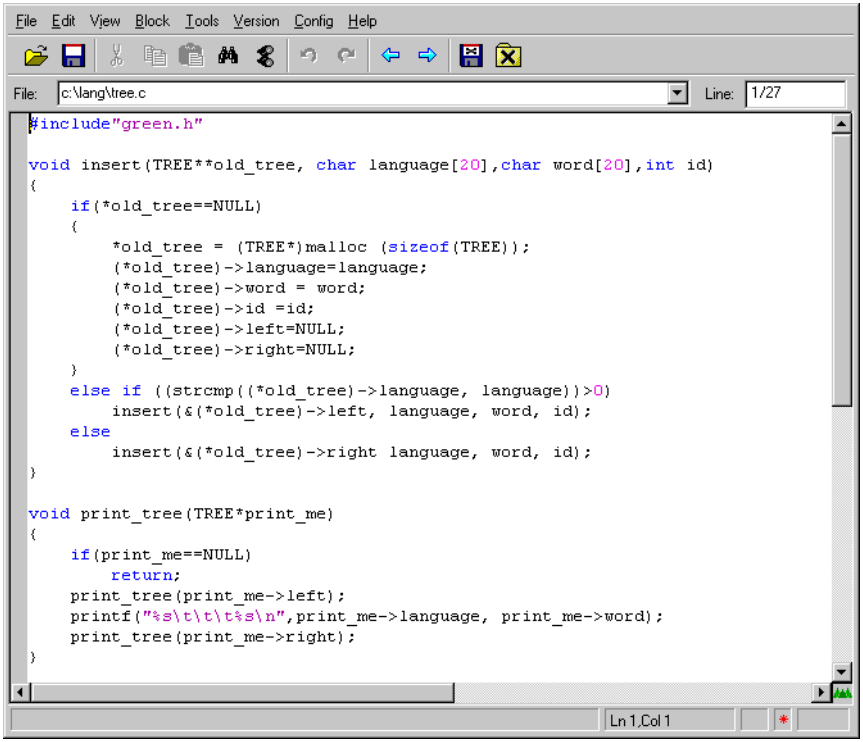
This chapter contains:

- The main Editor window
- Editor menus
- Editor toolbar
- Location fields
- Status bar
- Merge dialog boxes
- Search dialog box
- Goto dialog box
- Per File Settings dialog box
- File chooser
- Print dialog box

This chapter describes all buttons and menus used in the main Editor window.

The main Editor window

When you open the Editor, you see the following window:



The main menu bar at the top, rests on a tool bar of Editor buttons. Underneath the button bar is the title bar. Below this is the source pane, where your code to be edited is displayed. The options in the menu bar, the tool bar, and the title bar are explained below.

Editor menus

The following tables contain all menu items along with brief descriptions of the items. For each menu item, the equivalent command, if any, is provided for advanced users wishing to configure their menu settings. See Chapter 10, “Configuration commands” for more information.

File menu

NOTE: Spaces are not allowed in filenames. This restriction applies throughout the entire MULTI development environment.

File menu (editor)		
Menu item	**Meaning**	**Command**
New Editor...	Opens the Edit File dialog box, which you use to open a file in a new Editor window. To open an existing file, browse and select the file. To create a new file, enter the new filename and click Open. **Tip:** Use the wildcards '*' (any number of characters) and '?' (one character) to match and select multiple files.	LoadFile
Open...	Opens the Edit File dialog box, which you use to open a file in the current Editor window. To open an existing file, browse and select the file. To create a new file, enter the new filename and click Open. The file is pushed to the top of the context stack for the current window. **Tip:** Use the wildcards '*' (any number of characters) and '?' (one character) to match and select multiple files.	OpenFile
Save	Saves the current file.	Save
Save As...	Opens the Save As dialog box, which you use to save the current file under a different name.	SaveAs
Save All...	Opens a dialog box that lists all open files with changes that have not been saved. Select the check box next to the files you want to save, then click Save Selected. If you want to save all of the listed files, click Save All.	QuerySaveAll
Revert to Saved	Removes all changes made to the current file since the last time you saved it.	Revert
Print...	Opens the print dialog, which will allow you to print the current file to either a printer or a file. See "Print dialog box" on page 185 for more information.	Print
1, 2, 3, 4	Up to four previously viewed files may be listed. Select one to open it into the current Editor window.	OpenFile
Close File	Closes the top file in the Editor's file stack. If the file is not saved, it will prompt you to save it. If the file was checked out of version control this session, it will prompt you to check it back in.	Pop
Close Editor	Prompts you to save changes made to open files, then exits the Editor.	Close
Exit All	Prompts you to save changes made to open files, then quits all MULTI tools that are currently running.	Quit

Edit menu

Edit menu (editor)		
Menu item	**Meaning**	**Command**
Undo	Allows you to undo each of the changes made to the current file since it was opened.	Undo
Redo	Restores the edit that was just removed by selecting Undo. You can redo each undo, until a new change is made to the file.	Redo
RepeatLastEdit	Repeats the last edit you made to the file. For example, suppose you replace a selection with the word albatross. If you select a new part of the file and select Repeat Last Edit, the selection is also replaced with the word **albatross.** Only works for some types of edits. For more details, see “To repeat the last change you made to a file” on page 149.	RepeatLast
Cut	Copies the current selection to the clipboard and deletes selection from the current file.	Cut1
Copy	Copies the current selection to the clipboard.	Copy1
Paste	Pastes the clipboard contents in the current location.	Paste1
Delete	Deletes the current selection. If there is no current selection, this deletes the character after the insertion point.	Delete
Select All	Selects the entire contents of the current file.	SelectAll
Find...	Opens the Search dialog box. See also “Search dialog box” on page 178. **Tip:** To quickly search for a string without using the dialog box, use Ctrl+f.	Find
Goto...	Opens the GoTo dialog box, which you use to go to a new file, a line within the current file, or a function. See also “Goto dialog box” on page 182. **Tip:** To quickly goto a line number without using the dialog box, use ctrl+g.	Goto

View menu

View menu (editor)		
Menu item	**Meaning**	**Command**
Language	Select the programming language used in the current source file. The Editor uses this setting for syntax coloring, commenting, and auto-indenting features.	Language
Per File Settings...	A list of variables which can be set for an individual file for the current session only. See “Per File Settings dialog box” on page 182 for more information.	EditorFlags
Next File	Accesses the next open file in the Editor’s stack.	CyclePushBack
Previous File	Accesses the previous open file in the Editor’s stack.	CyclePush
Flash Cursor	Scrolls to and flashes the line containing the cursor.	FlashCursor
Match	Searches backward from the cursor for the first open parenthesis, square bracket, or curly brace at the same nesting level as the cursor. It then selects all the text enclosed by the corresponding close parenthesis, square bracket, or curly brace.	SelectToMatch
Column...	Puts the cursor at the specified column on the current line, if possible.	Column
Read Only	Toggles the file between read-only and writable modes. A dot next to this menu item indicates that the file is currently read-only.	ToggleReadOnly

Block menu

Block menu (editor)		
Menu item	**Meaning**	**Command**
Indent	Adds an indent at the beginning of the current line or selected lines. To set the size of the indent, choose Config > Options..., then select the Editor tab and edit the Indent size field. The default size is four spaces.	Indent
Unindent	Unindents the current line, deleting a number of spaces equal to or less than the size of an indent from the beginning of the current line. To set the size of the indent, choose Config > Options..., then select the Editor tab and edit the Indent Size field. The default size is four spaces.	Unindent
Auto Indent	Indents the current line or block of lines to positions indicated by the syntax of the code. Available for C, C++, and Ada languages. See “To let the Editor indent your code” on page 152 for more information.	AutoIndent
Comment	Inserts the appropriate characters to signify that the selected text is a comment, not code. The comment style that is used is determined by the language that you have set in View > Language.	CommentBlock
UnComment	Removes comment style characters from the selected text to make it active code.	UnCommentBlock
UpperCase	Changes all the characters in the current selection to upper case.	UpperCaseBlock
LowerCase	Changes all the characters in the current selection to lower case.	LowerCaseBlock
Rect Copy	Copies a rectangular subsection of the current selection to the clipboard. For more details, see “To copy a column of text” on page 149.	RectCopy1
Rect Cut	Copies a rectangular subsection of the current selection to the clipboard, then deletes it. For more details, see “To copy a column of text” on page 149.	RectCut1
Rect Paste	If you have used Rect Copy or Rect Cut to copy a selection to the clipboard, you can use Rect Paste to put the selection into the file without linebreaks. If you use Edit > Paste to paste a rectangular selection, linebreaks will be added.	RectPaste1

Block menu (editor)		
Menu item	**Meaning**	**Command**
Cut Lines	Extends the current selection to the closest line boundaries, copies the resulting selection to clipboard number two, then deletes it.	SelectToLines; Cut2
Join Lines	Joins two lines of text by removing the new line character from the end of the current line, as well as all initial whitespace on the next line, then adding one space.	JoinLines
Insert File...	Opens the Insert dialog box, which you use to insert the contents of a separate file into the current file. The contents of the selected file is placed on the line above the cursor.	InsertFile

Tools menu

Tools menu (editor)		
Menu item	**Meaning**	**Command**
Insert Date	Inserts the current date, as a formatted string, at the cursor's position in the current file.	Date
Grep...	Searches for a regular expression in all open files. (If you have a debugger open when you run this command, it will also search in all of your program's source files.) The output from this command appears in a temporary Editor window. Double click any of the lines in the temporary window to open a new Editor on the specified file. See also "Grep" on page 205.	Grep
Make...	Calls the **make** utility on a project of your choosing.	Make
Execute Shell Command...	Executes a command to the sh shell. Output will appear at the insertion point in the currently open file.	ExecuteCmd
Command to Window...	Executes a command to the sh shell. Output will appear in a new temporary Editor window.	CommandToWindow
Notepad...	Calls a small Editor window on a scratch file.	Notepad
Execute Editor Commands...	Prompts for an editor command to execute in the current editor. See Chapter 7, "Editor commands" for valid commands.	MiniBuffer

Tools menu (editor)		
Menu item	**Meaning**	**Command**
Append TagFile...	Allows you to specify a tag file (in the format determined by the ctag utility) that the Editor can use to find procedures in files. By default, the Editor looks for a file called "tags". See the command “OpenTag” on page 208 for more information.	AppendTagFile
Reset Tags...	Resets the tag file the Editor uses back to the default of "tags".	ResetTags
Merge Files...	Merges either two or three files. See “Merging files” on page 156 for more details.	MergeFiles
Diff Files...	Finds and displays the differences between two files. See also “Comparing files” on page 160.	DiffFiles

Version menu

Version menu (editor)		
Menu item	**Meaning**	**Command**
Check Out	Checks out the current file from version control for editing. Enabled only if the current file uses version control.	Checkout
Check In	Checks the current file back into version control, making the file read-only. Enabled only if the current file uses version control.	Checkin
Check In All	Checks in all of the files currently checked out in the Editor.	QuerySaveCheckinAll
Discard Changes	Reverts the file back to the last checked in version from version control. Enabled only if the current file uses version control.	Discard
Place Under VC	Puts the current file under version control. Once a file is placed under version control, the file must be checked out before changes can be made.	CreateLog
Auto Checkout	Toggles Auto-Checkout mode. When selected, the Editor automatically checks out a file from version control when you start to make changes. If deselected, you must manually check out a file that uses version control before making changes.	AllowAutoCheckout

Version menu (editor)		
Menu item	**Meaning**	**Command**
Show History...	Shows information about all versions of the current file with a dialog box that allows you to open any version in a new Editor. Enabled only if the current file uses version control.	ShowHistory
Show Last Edit	Finds the version of the current file that changed the selected text. This command opens a window on the version that changed the text, placing the cursor at the beginning of the change. Enabled only if the current file uses version control, and is checked in (if the current file is checked out, the results of this operation is undefined). See also “ShowLastEdit” on page 210.	ShowLastEdit
Revert To History...	Displays a window with a list of all versions and comments. You can choose a version from this list to load into the Editor, replacing the currently open version. Enabled only if the current file uses version control.	RevertHistory
Revert To Date...	Displays a small window to enter a version date. The Editor loads the version current to the specified date. Enabled only if the current file uses version control.	RevertDate
Revert To Version...	Displays a window to enter a version number to load into the Editor. Enabled only if the current file uses version control.	RevertVersion

Config menu

Config menu (editor)		
Menu item	**Meaning**	**Command**
Options...	Displays the Options dialog box, which you use to change options that affect the way the Editor and other MULTI tools look and behave.	configoptions
Save Configuration as Default	Allows you to permanently save the changes you made in the Appearance Settings and Functionality Settings dialog boxes.	SaveConfig
Clear Default Configuration...	Clears all saved changes and reverts to all default Appearance and Functionality settings dialog boxes.	ClearConfig
Save Configuration...	Save the Appearance and Functionality settings to a user specified file.	SaveConfigToFile
Load Configuration	Load Appearance and Functionality settings from a user specified file.	LoadConfigFromFile

Help menu

Help menu (editor)		
Menu item	**Meaning**	**Command**
Editor Help...	Opens MULTI's help index.	Help
Manuals	Opens the "Manuals sub-menu", which will display a list of manuals appropriate to your version of MULTI. Choosing one of these manuals will open the online help to the first page of that manual.	n/a
Identify...	Displays help for the next key or mouse click sequence that you perform.	Identify
About MULTI...	Opens the About banner.	About

Right-click pop-up menu

Menu item	Meaning	Command
Cut	Copies the selection to the clipboard and deletes selection from the current file.	Cut1
Copy	Copies the selection to the clipboard.	Copy1
Paste	Pastes the clipboard contents in the current location.	Paste1
Undo	Allows you to undo each of the changes made to the current file since it was opened.	Undo
Jump to Function	Goes to the location of the function selected (if a selection exists), otherwise to the one indicated in the menu item	OpenTag

Menu item	Meaning	Command
Comment	Inserts the appropriate characters to signify that the selected text is a comment, not code. The comment style that is used is determined by the language that you have set in View > Language. This menu item appears only for context menus brought up on selections.	CommentBlock
Uncomment	Removes comment style characters from the selected text to make it active code. This menu item appears only for context menus brought up on selections.	UnCommentBlock
Auto Indent	Indents the current line or block of lines to positions indicated by the syntax of the code. Available for C, C++, and Ada languages. See “To let the Editor indent your code” on page 152 for more information.	AutoIndent

Editor toolbar

Editor toolbar		
Button	**Meaning**	**Command**
	Opens a file into the current Editor window.	OpenFile
	Saves the current file.	Save
	Copies the current selection to the clipboard, then deletes it.	Cut1
	Copies the current selection to the clipboard.	Copy1
	Pastes the contents of the clipboard.	Paste1
	Opens the Editor’s search window. See “Search dialog box” on page 178.	Search
	Opens the GoTo dialog box, which you use to go to a new file, a line within the current file, or a function. See “Goto dialog box” on page 182 for more information.	Goto
	Undoes the last edit. You can undo to the original status of your file.	Undo

Editor toolbar		
Button	**Meaning**	**Command**
	Redoes the last edit that was undone. You can redo all undos until a new edit is made.	Redo
	Accesses the previous open file in the Editor's stack.	CyclePush
	Accesses the next open file in the Editor's stack.	CyclePushBack
	Quits after saving permanent files. Temporary files like the progress window (build output) and notes are not automatically saved.	Done
	Closes the current file. You will be prompted to save and/or check in the file before closing it.	Pop
	Quits the Editor. You will be prompted to save and/or check in all edited files before quitting. You can configure whether or not to have this button. See also "Display close (x) buttons" on page 243.	Close

Location fields

Below the toolbar are two text fields that control and describe where you are in the current file. These fields display the following information:

File:

The **File:** field contains the name of the file you are currently editing. If you want to edit another file, enter another filename in the **File:** field and press Enter. If the filename you entered does not exist, MULTI prompts to create a new file. The file will appear in the current window, pushing the current file onto the Editor's stack.

To open the file in a new Editor window, use Shift+Enter instead of Enter.

Line:

The **Line:** field tells you what program line your cursor is on. You can go to a specific line by clicking this text field and typing a new line number.

Status bar

The status bar is at the very bottom of the Editor window. It displays the following information:

Status box

The left corner of the status bar displays status, usage, and error messages. When the mouse hovers over certain widgets, this area will display a usage message, and when the source pane is active it will display status and error messages. For example, when you press Ctrl+f, the left corner of the Status Bar displays the search text as you type it in, and when you type Ctrl+g it displays the line number to goto as you type it.

Cursor position indicator

Displays the current line and column on which the text cursor resides.

Read-only indicator

When the current file is read-only, a Stop Sign displays in the Status Bar.

Change dot

If changes were made to the file since the last time it was saved, a small red star appears in the bottom right corner of the screen. When the file is saved, this star disappears.

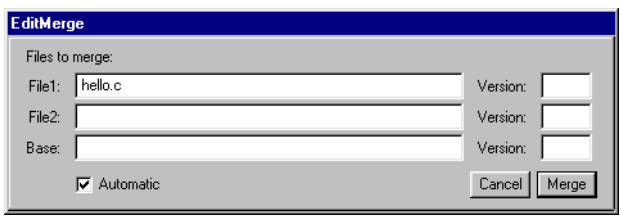
Version control status

If the current file is controlled by version control, **VC** appears in the right corner of the Status Bar. If you have the current file checked out from version control, the letters will be red. Otherwise they will be black.

Merge dialog boxes

Merge dialog box

(Tools > Merge Files...)



If you are using a version control system and want to merge a different version of a file from disk, enter the filename in the Filename field and the version of that file to be merged in the adjacent Version field.

File1

Enter the name of the first file you want to merge.

File2

Enter the name of the second file you want to merge. If you specify the same file as you specified in the File1 field, then the File1 file refers to the copy currently open in the Editor, while File2 file refers to the file on disk. This is useful if someone else has edited the file at the same time you were working on it.

Base

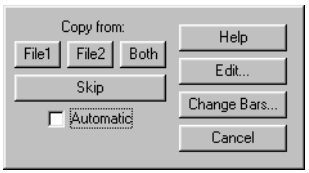
Enter the name of the file from which the other two files, File1 and File2, are derived. If you are merging two files, then leave this field blank

Automatic

Select if you want the Editor to try to resolve merges without prompting you. Deselect if you want to manually review every proposed merge. This field has no meaning when merging two files.

Control panel (two-file merge)

(Tools > Merge Files... ; fill in File1 and File2, and click Merge.)



This dialog box allows you to control the merge. The Editor pauses at each difference it finds and waits for you to tell it what to do.

The panel contains the following buttons:

Skip

Finds the next difference.

Help

Opens help on the control panel.

Cancel

Aborts the merge, closing all merger windows.

File1

Copies the selected text from File1.

File2

Copies the selected text from File2.

Both

The first time you press this button, a dialog box appears asking you how you want to merge the two selections: in what order, with change bars around them, with comments in front, between, or after them, and so forth. The next time you press this button, the last values entered are used.

Change Bars...

Changes the way the All button works, and opens the same dialog box as the first time you press the All button.

Edit...

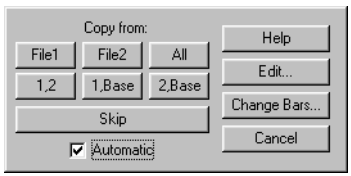
This performs the same action as **1,2**, but then allows you to modify the changes in a temporary editor before merging them into the new file.

Automatic

Enables or disables the automatic merge feature. If enables, the Editor will try to make merge changes without prompting you. If disabled, the Editor will let you manually control every merge change.

Control panel (three-file merge)

(Tools > Merge Files... ; fill in File1 and File2 and Base, and click Merge.)



This dialog box allows you to control the merge. If the automatic feature is turned off, you can control the entire merge. If the automatic feature is turned on, you only control the merge of conflicting changes with this window. The panel contains the following buttons:

Skip

Finds the next difference.

Help

Opens help on the control panel.

Cancel
Aborts the merge, closing all merger windows.

File1
Copies the selected text from File1.

File2
Copies the selected text from File2.

All
Copies the selected text from all three files. The first time you press this, or any of the next three buttons, a button dialog box will appear asking you how to merge the selection: in what order, with change bars around them, with comments in front, between, or after them, and so forth. The next time you press this button, it will use the last values entered.

1,2
Merges the selections from File1 and File2.

1,Base
This is identical to **1,2** except it merges the selections from File1 and the base file.

2,Base
This is identical to **1,2** except it merges the selections from File2 and the base file.

Change Bars...
Changes the way **1,2**, **1,Base**, **2,Base**, and **All** buttons work, and opens the same dialog box as the first time you press any of those buttons.

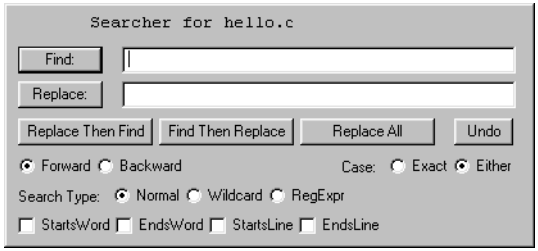
Edit...
This performs the same action as **1,2**, but then allows you to modify the changes in a temporary editor before merging them into the new file.

Automatic
Enables or disables the automatic merge feature. If enables, the Editor will try to make merge changes without prompting you. If disabled, the Editor will let you manually control every merge change.

Search dialog box

To open the search dialog box, do one of the following:

- Click the Search button ().
- Choose Edit > Find...
- Press Ctrl+Shift+f



This dialog box searches and replaces text in your file. You can open a search dialog box for each file you are editing.

Enter the desired text or search string in the field next to the Find button. To search for control characters, such as a tab, use the Quote command to enter them correctly (by pressing Ctrl+\). For example, if you want to search for a tab, press Ctrl+\ and then press Tab. You can copy and paste special characters from an Editor window.

To replace text, enter the desired text or replace string in the field next to the Replace button.

The Editor searches from the current location in the file towards the end of the file for a forward search, and toward the top of the file for a backward search. If the search string is not found before it reaches the end or the beginning, it prints a message and stops. If you start again, it resumes the search from the beginning or the end of the file.

There are six buttons in the search window:

Find
: Searches for and highlights the next occurrence of the search string. Simply pressing Enter also searches for the next occurrence.

Replace
: Replaces the current selection with the replace string.

Replace Then Find
: Replaces the current selection and then searches again.

Find Then Replace
: Searches for the next occurrence of the search string, and replaces it with the replace string if found.

Replace All
: Starts at the beginning of the file and replaces all occurrences of the search string with the replace string.

Undo
Undoes the last Editor command.

There are a number of check boxes, which click on and off, and several radio buttons, small circles that either contain a solid dot for ‘on’ or are empty for ‘off.’ Radio buttons are in sets, and only one turns on at a time. The check boxes and radio buttons include:

Forward or Backward
Determines whether the search proceeds forward or backward.

Case: Exact or Either
Determines whether case should be matched. If **Exact** is on, then only strings that exactly match the case are found. For example, **Fly** matches **Fly**, but not **fly** or **FLY**. If **Either** is on, then case is ignored. For example, **Fly** matches both **fly** and **FLY**.

StartsWord or EndsWord
If only **StartsWord** is on, then the search string must appear at the beginning of a word. For example, **fly** matches **fly** or **flybat**, but not **batfly**.

If only **EndsWord** is on, then the search string must appear at the end of a word. For example, **fly** matches **fly** or **batfly**, but not **flybat**.

If they are both on, then the string must form a complete word. For example, **fly** matches **fly**, but not **flybat** or **batfly**.

If neither is on, then any occurrence of the string is found.

StartsLine or EndsLine
These are similar to **StartsWord** and **EndsWord** above, except they apply to the beginning and end of a line.

Normal
If this is on, then there are no special characters; that is, characters only match themselves.

WildCard
If this is on, then the following characters have a special meaning in the search string:

? Matches any single character except a newline.

***** Matches any number of characters except newlines.

RegExpr
If this is on, then the following characters have a special meaning in the search string. In this description, a regular expression is an expression

using any combination of the following special characters. Note that you cannot match a newline.

Regular expressions	
.	(A period) Matches any single character except a newline.
[string]	Matches any single character appearing in the *string*. For example, **[abc]** matches an **a**, **b**, or **c**. You can specify character ranges by separating the start and end of the range with a **-**. For example, **[b-e]** matches any character between **b** and **e** (**b**, **c**, **d**, and **e**). To include a **]** as part of the *string*, make it either the first character of the *string*, or the last character of a range. For example, **[]abc]**.
	If the first character of the *string* is a **^**, then it matches any character that does not match the rest of the *string*.
^	At the start of the search string, this matches the beginning of a line.
$	At the end of the search string, this matches the end of a line.
<	At the start of the search string, this requires the rest of the search string to match the beginning of a word. Same as the **StartsWord** toggle.
>	At the end of the search string, this requires the rest of the search string to match the end of a word. Same as the **EndsWord** toggle.
(re)	Matches the regular expression *re* enclosed in parentheses.
re*	Matches zero or more occurrences in succession of the regular expression *re*.
re1|re2	Matches regular expression *re1* or regular expression *re2*.

For example:

a.d matches **and**, **a d**, and **aud**.
a.*d matches **ad**, **are d**, and **abd**.
<and matches **and**, but not **stand**.
are|is matches either **are** or **is**.
(are|is)* bad matches **are bad**, **is bad**, **areisare bad**, and **bad**.

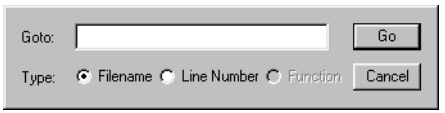
The settings in this dialog box set the defaults for the next quick search.

Goto dialog box

To open the Goto dialog box, do one of the following:

- Click the Goto button ( ).
- Choose Edit > Goto...
- Press Ctrl+Shift+g

You can use this dialog box to ‘goto’ a file, line number, or function by selecting the appropriate radio button.



Goto a file

When Filename is selected, you can type a filename into the textfield to open it into the current editor.

Goto a line number

When Line Number is selected, you can type a number into the textfield to go to that line in the current editor.

Goto a function

When Function is selected, you can type a function name into the textfield to search for a function name and open its file into the current editor. This option is only available if a **ctags** style tag file named "tags" resides in the current directory, or if a tags file has been specified using the Tools > Append TagFile... menu option or by using the **AppendTagFile** editor command. See **AppendTagFile on page 208** and/or **ResetTags on page 209** for more information.

Per File Settings dialog box

The Per File Options dialog box lists a number of indenting and text wrapping options that you can set for the current file and current session only.

To open this dialog box, choose View > Per File Settings.

Indent size

Controls the tab size that the Editor inserts when you press the Tab key. The default is 4 spaces. If you enter 0 to disable tabs, pressing Tab will insert a single space.

Ada indent size

For Ada language file types only. Same as above, except the default is 3. See “Language” on page 167 for more information.

Ada continuation size

For Ada language file types only. Determines how far a line of Ada source code is indented if it continues to multiple lines.

Wrap column

If word wrap is enabled, this specifies the column at which word wrap takes affect. See “Word wrap” on page 183 for more information. If a line grows to longer than this number of columns as you are typing, the Editor will insert a page break at the first white space before the column. It will indent the newly formed line by the same amount as the line above it, plus the wrap indent offset, described below.

Wrap indent offset

If word wrap is enabled, this determines how much the wrapped line will be indented past the indention of the line above it. See “Word wrap” on page 183 for more information.

Word wrap

This check box enables or disables word wrap. If word wrap is enabled, the Editor will not wrap long lines while loading a file, but will wrap lines automatically as you type. See Wrap column and Wrap indent offset (above).

Disk format

Determines whether the file will be written in UNIX format or DOS format when the Editor saves it to disk. It will default to the appropriate setting.

File chooser

This File Chooser Window allows you to browse and choose files for various functions.It displays the following information:

Directory

This textfield displays the current directory being listed. Type in a new directory name and press return to display a different directory list.

Directory Buttons

This set of buttons allows you to jump quickly to different important directories:

Button	Meaning
	Pops up to the parent of the current directory.
	Jumps to the Current Working Directory.
	Jumps to the directory MULTI is running from.
	Jumps to the user's home directory.

File List

Below the directory text field is the file list. Double click on a directory to enter it. Double click on a filename to choose it and dismiss the window. To sort the list in ascending or descending order by any column, click on the desired column header. Resize any of the columns by clicking and dragging the column separators in the column headers. If multiple files are allowed for the present operation (i.e. Edit and Open), the File Chooser will allow you to select multiple files with the mouse. Hold down the Control key to select non-consecutive files. Use the Shift key to extend a consecutive list of selected files.

Filename

Type a filename or directory name into this textfield. The selected file in the file list will change as you type in this field to reflect the closest match. If you enter a directory name and press return or enter a trailing slash, the file list will change to that directory. If you type a filename and press return the file will be chosen and the window dismissed. Chooser will translate the wildcards '*' and '?' to match respectively: 1) any number of consecutive characters, and 2) one character.

Action buttons

There are two buttons in the lower right corner of the file chooser window. The upper button displays the action that takes place upon pressing it (i.e. Edit or Save). The lower button is the Cancel button, which closes the window without taking any action.

Print dialog box

The Print menu item opens a Print dialog box. You can use this dialog box to print the current file in various ways. The following describes the items in the dialog box.

Print To

The Print To radio button allows you to choose to print to a printer or to a postscript file.

Print Command

For Print To Printer mode only, this textfield displays the actual command that will be run when the Print button is clicked. Use this to add printer or system specific options or commands.

Filename

For Print To File mode only, this textfield displays the postscript file that will be written to when you click the Print button. You can use the Browse... button to look for a file to print to.

Font Name

You can use this combo box to pick the font that will be used when you click the Print button.

Font Size

Use this combo box to select the font size that will be used when you click the Print button.

Paper Size

Use this radio button to select the paper size that will be used when you click the Print button.

Orientation

Use this radio button to select the paper orientation that will be used when you click the Print button.

Columns

Use this radio button to select the number of columns that will be used when you click the Print button.

Print button

When all options are set correctly, click this button to print the file.

Chapter 7

Editor commands

This chapter contains:

- Navigation commands
- Indentation commands
- Drag-and-drop commands
- Text deletion commands
- Clipboard commands
- Block commands
- Search commands
- Undo/Redo commands
- File commands
- Tool commands
- Tag commands
- Version control commands
- Configuration commands
- Help commands
- Insert commands
- ’if’ conditional commands

Most of the commands listed in this chapter duplicate actions you can perform using the Editor's Graphical User Interface (GUI). Green Hills Software provides these commands so you can customize the Editor to help you work more efficiently. For example:

- If you do not want to use the mouse when you are editing, you can bind any command to a keystroke combination. Since all editor actions available from the GUI are also commands, all Editor functionality may be bound to keys in this way.
- If you find that you are often performing the same two or more actions in succession, you can combine the commands for these actions into a single keystroke, key sequence, mouse button combination, GUI button, or menu item.

Please see Chapter 9, "Configuring and customizing MULTI" to find out how you can bind commands to menus, keystrokes, mouse clicks, and buttons. Of course, many of these commands are already accessible via the default bindings; for a list of those defaults, please see Chapter 8, "Default key bindings".

Navigation commands

The cursor is the point in the open file where new text appears in response to keystrokes, and is denoted by a flashing vertical bar between two characters. These commands change the location of the cursor, scrolling the open file as necessary to keep the cursor within the Editor's window. None of these commands will modify the contents of the open file in any way.

Up

Moves the cursor up one line.

Down

Moves the cursor down one line.

Left

Moves the cursor one character to the left. If the cursor is already in the first column of the line, this command has no effect.

Right

Moves the cursor one character to the right. If the cursor is already in the last column of the line, this command has no effect.

PageUp

Scrolls the window and moves the cursor up one window-length.

PageDown

Scrolls the window and moves the cursor down one window-length.

UpSome

Moves the cursor up by *some* lines. The size of the “some” parameter defaults to 5, but may be changed via the *Ctrl-cursor jump size* field, which is located in Config > Options... > Editor tab.

DownSome

Moves the cursor down by *some* lines. The size of the "some" parameter defaults to 5, but may be changed via the *Ctrl-cursor jump size* field, which is located in Config > Options... > Editor tab.

LeftSome

Moves the cursor to the left by *some* characters. The size of the "some" parameter defaults to 5, but may be changed via the *Ctrl-cursor jump size* field, which is located in Config > Options... > Editor tab.

RightSome

Moves the cursor to the right by *some* characters. The size of the "some" parameter defaults to 5, but may be changed via the *Ctrl-cursor jump size* field, which is located in Config > Options... > Editor tab.

LeftU

Moves the cursor one character to the left. If the cursor is in the first column of the line, moves the cursor to the last column of the previous line.

RightD

Moves the cursor one character to the right. If the cursor is in the last column of the line, moves the cursor to the first column of the next line. If the cursor is in the last column of the last line of the open file, this command has no effect.

Return

Moves the cursor to the first column of the next line.

Word

Moves the cursor to the end of the next word. If the cursor is currently not on a word (a group of consecutive alphanumeric characters), this command moves the cursor to the end of the next word in the open file.

ReverseWord

Moves the cursor to the beginning of the previous word, in a similar fashion to the **Word** command.

SOL

Moves the cursor to the first non-whitespace character on the line. If the cursor is already to the left of the first non-whitespace character, moves the cursor to the first column of the current line.

EOL

Moves the cursor to the last column of the current line.

SOF

Moves the cursor to the first column of the first line of the open file.

EOF

Moves the cursor to the last column of the last line of the open file.

SOL0

Moves the cursor to the first column of the current line (before any indentation), even if the first character is indented.

SOL1

Moves the cursor to the first non-whitespace character on the line, even if the cursor is currently to the left of the first non-whitespace character.

EditLine

Allows you to enter the line number you want to place the cursor on. When you run the editline command, the prompt “Goto Line:” is displayed in the status bar. Enter the line number.

If you prefer to enter a new line number in a dialog box, see the **Goto** command.

If you prefer to enter a line number as a parameter, see the **LineD** command.

Goto

Opens the Goto dialog box that allows you to specify a filename, line number, or tag to open in the editor.

LineD

Format: **LineD** [*line_number*]

Moves the cursor to the specified line. If no parameter is specified, this command opens a dialog box that allows you to specify the line to which the cursor should be moved.

Column

Format: **Column** [*column_number*]

Moves the cursor to the specified column in the current line. If no parameter is specified, this command opens a dialog box that allows you to specify the column to which the cursor should be moved.

FlashCursor

Flashes the line the cursor is on. This command also turns off insert mode off. (See the **EnterInsertMode** command.) This command is equivalent to View > Flash Cursor.

Indentation commands

Indentation is whitespace at the beginning of each line, and may be used to more clearly denote the hierarchical structure of your code, thus making it more readable. Each of these commands change the way that the current line is indented, either by altering the indentation manually or by automatically applying language-specific heuristics to determine how the line should be indented. If a number of lines are selected, these commands will operate on each line in the selection instead.

Indent

Adds an indent at the beginning of the current line or selection. The default indent size is 4 spaces, which can be overridden on a per-file, per-session basis with the **EditorFlags** Editor command. The default indent size is specified in Config > Options... under the Editor tab.

Unindent

Removes an indent from the beginning of the current line or selection. Up to one indent worth of whitespace is removed. If there is less whitespace than the indent size at the beginning of the current line, this command will remove all of it. If there is no whitespace at the beginning of the current line, this command has no effect. See the **Indent** command for notes about the indent size.

SelectLanguage

Selects the language to use for color syntax and for auto-indenting. Available modes include: C, C++, Ada, Pascal, Fortran, None, GreenHillsScript.

Note: The C and C++ syntax highlighting module attempts to gray out any code that is enclosed by the `#if 0` preprocessor directive. However, if several such directives are nested, only the outermost one will be highlighted correctly.

AutoIndent

For C, C++, and Ada only. Automatically indents the current line or selection according to the structure of the code. Several configuration options affect the operation of **AutoIndent**. See “Auto Indent Options” on page 255 for more information.

AutoIndentImplicit

Like AutoIndent, except that it can be turned off (made to do nothing) by setting the configuration option **AIimplicitindent** to *Off*. This is equivalent to clearing the Implicit Auto Indent check box, which is in Config > Functionality Settings... under the Editor tab. See “Implicit auto indent” on page 255 for more information.

AutoIndentOrTab

Like **AutoIndent**, except that if the current file is in a language not supported by AutoIndent, a Tab is inserted instead. This command is typically bound to the Tab key.

Selection commands

The current selection is a highlighted area of the text in the open file. When text is selected, many commands operate differently upon the selected text than they

would ordinarily. These differences are documented in the individual command descriptions. The MULTI Editor supports two selections:

- Text that is part of the *primary selection* can be manipulated by many commands, such as the Clipboard Commands, Indentation Commands, and Drag-and-Drop Commands. The primary selection is replaced by any typed or inserted text, and is cancelled by any Navigation Command. One end of the primary selection, the *cursor-end*, is always at the cursor's current location, and may be either the start or the end of the selection. By default, a primary selection is created by dragging over text with the left mouse button.
- Text that is part of the *secondary selection* can be replaced using either the **SecondarySelectionReplace** or **SecondarySelectionReplaceClip** commands. Any other non-selection-related MULTI Editor command or keystroke will cancel a secondary selection if it exists. By default, a secondary selection is created by dragging over text with the middle mouse button. When the middle button is released, **SecondarySelectionReplace** is executed.

The Selection Commands can be used to create or cancel these two types of selections. Commands which don't explicitly mention the secondary selection operate on the primary selection only. With the exception of the **SecondarySelectionReplace** and **SecondarySelectionReplaceClip** commands, none of the commands in this section will modify the contents of the open file in any way.

NoSelection

Moves the cursor to the beginning of the current primary selection and cancels it. If no primary selection exists, this command has no effect.

SelectAll

Makes the entire open file the primary selection, moving the cursor to the end of the open file. The window is not scrolled to show the cursor.

SelectWord

Makes the current word the primary selection, moving the cursor to the end of the word. The window is not scrolled to show the cursor.

SelectLine

Makes the entire current line (including any indentation) the primary selection, moving the cursor to the end of the line. The window is not scrolled to show the cursor.

SelectMatch

If the character immediately after the cursor is a *paired character*, such as a parenthesis, square bracket, quote, or curly brace, it will select the corresponding paired character in the open file. For example, if the cursor is positioned immediately before a closing parenthesis, this command will select the opening parenthesis character that matches it. If the cursor is not over one of these characters, or if the parenthesis, bracket, etc. has no match, this command will cause a beep and have no further effect. See also **SelectToMatch**.

SelectToLines

Extends the current selection so that it completely includes the first and last lines of the current selection. The new selection begins at the first column of the first line of the current selection and ends at the last column of the last line of the current selection. If there is no current selection, this command is equivalent to **SelectLine**.

SelectToMatch

Extends the selection to include the nearest paired characters (quotes, parentheses, etc.) that enclose the current selection, along with any text that is between them. If there is no selection, this command makes the selection include the nearest paired characters on either side of the cursor. This command is equivalent to View > Match. See also **SelectMatch**.

ContinueSelection

Moves the cursor-end of the current selection with the next Navigation Command, extending or shrinking the selection accordingly. For example, if the cursor-end of the selection is last and a **Down** command is preceded by **ContinueSelection**, the cursor will move down and the selection will be extended down one line to the cursor's position. However, if the cursor-end of the selection is first (at the beginning of the selection) and a **Down** command is preceded by **ContinueSelection**, the cursor will move down and the selection will shrink by one line to the cursor's position.

SelectionStart, SelectionGrab, SelectionExtend, SelectionAdjust

These commands create and manipulate selections in a way that is dependent upon the mouse. To be useful, these commands should be bound to mouse buttons and used in conjunction with one another. (See “Default mouse settings” on page 225 for a sense of how these commands are used.)

SecondarySelectAll, SecondarySelectLine, SecondarySelectWord

These commands have the same behavior as their primary selection counterparts, except that they operate on the secondary selection.

SecondarySelectionStart, SecondarySelectionExtend, SecondarySelectionAdjust

These commands have the same behavior as their primary selection counterparts, except that they operate on the secondary selection.

SOLSecondary

Starts the secondary selection at the beginning of the current line, the first column, and changes it to zero length. It contains no characters.

SecondarySelectionReplace

Deletes the text in the secondary selection, replacing it with a copy of the text in the primary selection. This command also cancels the secondary selection.

SecondarySelectionReplaceClip

Deletes the text in the secondary selection, replacing it with a copy of the text in the clipboard. This command also cancels the secondary selection. For more information about the clipboard, see “Clipboard commands” on page 197.

Drag-and-drop commands

The MULTI Editor supports editing the open file by dragging selected text around to move it. These commands perform various functions associated with this drag-and-drop behavior. Because they are all dependent upon the location of the mouse pointer, these commands are rarely useful when not bound to mouse buttons and actions. See “Default mouse settings” on page 225 for a sense of how these commands are used.

SelectionStartDrag

Starts a drag-and-drop operation. When the mouse is over a legal drop spot, the cursor will change into the **drop** cursor, allowing you to drop the text to a new location. At the completion of a drag-and-drop operation, the text will be deleted from the current location, and pasted to the dropped location. If there is no primary selection, this command has no effect.

By default this command is executed by left-clicking and dragging from within the current primary selection. If the control key is pressed during any point during a drag-and-drop operation, the operation will turn into a drag-and-drop-add operation.

Warning: This command must be followed by a **SelectionDrop**. See also **SelectionDrop** and **SelectionStartDragAdd**.

SelectionStartDragAdd

Starts a drag-and-drop-add operation. When the mouse is over a legal drop spot, the cursor will change into the **drop-add** cursor, allowing you to copy the text to a new location. At the completion of a drag-and-drop-add operation, the text will be copied to the dropped location. If there is no primary selection, this command has no effect.

By default this command is executed by left-clicking and dragging selected text, while holding down the control key. If the control key is pressed during any point during a drag-and-drop operation, the operation will turn into a drag-and-drop-add operation.

Warning: This command must be followed by a **SelectionDrop**. See also **SelectionDrop** and **SelectionStartDragAdd**.

SelectionDrop

This command will complete the drag-and-drop or drag-and-drop-add operation. Legal drag spots are anywhere in the editor text pane, except on the current selection, and are indicated by the mouse cursor changing into the **drop** (or **drop-add**) cursor. Illegal drop spots are indicated by the **no** mouse cursor. See **SelectionStartDrag** and **SelectionStartDragAdd** below.

By default this command will be called whenever the left mouse button is released during a drag-and-drop or a drag-and-drop-add operation.

Text deletion commands

These commands destroy text in the open file, either at the cursor or in the primary selection. Once text is destroyed by these commands, it can only be recovered with the undo command. (See “Undo/Redo commands” on page 201 for more information.)

Backspace

Deletes the text in the current primary selection and cancels the primary selection. If there is no selection, this command deletes the character immediately before the cursor.

Delete

Deletes the text in the current primary selection and cancels the primary selection.

Clipboard commands

- The *clipboard* is a special place set aside in your computer’s memory that may be used to store text for you to use later. Although it is not visible, the data will persist until you restart MULTI or until you replace it with different data. In fact, MULTI provides 4 different areas in which you may store text, which is useful for quickly moving or duplicating parts of the open file. These commands provide you with facilities for storing and retrieving text from the clipboard.

Copy1, Copy2, Copy3, Copy4

Copies the text in the current selection into the first and second, third, and fourth clipboards, respectively. The previous contents of the given clipboard will be lost after this command is issued, although all of the other clipboards will be unaffected.

Cut1, Cut2, Cut3, Cut4

Copies the text in the current selection into the first, second, third, and fourth clipboards, respectively, and then deletes the text in the selection from the open file. The previous contents of the given clipboard will be lost after this command is issued, although all of the other clipboards will be unaffected.

Paste1, Paste2, Paste3, Paste4

Inserts the text from the first, second, third, and fourth clipboards, respectively into the open file at the cursor's current location. The contents of the given clipboard will still be intact after this command is issued, so subsequent **Paste** commands will still insert the same text that was originally copied or cut (with the **Copy** or **Cut** command).

RectCopy1

Copies a rectangular subsection of the current selection to the first clipboard. If the selection extends across multiple lines, only the characters in the columns that are between the start and end of the selection will be stored in the clipboard.

For examples, see "To copy a column of text" on page 149. This command is equivalent to Block > Rect Copy.

RectCut1

Copies a rectangular subsection of the current selection to the first clipboard, and then deletes the copied text. If the selection extends across multiple lines, only the characters in the columns that are between the start and end of the selection will be stored in the clipboard. The remaining text in the selection will shift to the left to fill the space left by the text that was cut.

For examples, see "To cut a column of text" on page 150. This command is equivalent to Block > Rect Cut.

RectPaste1

If the first clipboard contains a rectangular section (that was put there by a **RectCopy1** or **RectCut1** command), inserts the selection as a rectangle starting at the current cursor position. Text on lines below the cursor will be shifted to the right to make room for the rectangle that is being inserted.

For examples, see "To paste a column of text" on page 150. This command is equivalent to Block > Rect Paste.

Block commands

These commands modify text in the current primary selection in convenient ways. Although they are designed to operate on selected text, most of these commands have some default behavior that occurs even when there is no selection.

CommentBlock

Comments the text in the selection using language-specific commenting (see also SelectLanguage). For languages such as C that do not support parenthesis-style nested comments, any comment-delimiting characters in the selected text will be mangled so that they are no longer parsed as comment characters. For example, if the following C code is selected:

```
int i=5; /* index */
void bad(void) {}
```

Then the **CommentBlock** will replace it with:

```
/*int i=5; /@*$ index $*@/
 - void bad(void) {}*/
```

In other words, the block would be surrounded with C-style comments '/*', '*/', nested comments would have '@' and '$' inserted to allow for uncommenting and correct nesting, and new lines are replaced by ' - '. Note that if these comments are now included in another CommentBlock command, they will change to '/@@*$' and '$*@@/', so subsequent UnCommandBlock commands will work as expected.

For C++, C++ style comments '//' will be prepended to every line.

Pascal behaves like C, except that the block is surrounded with '{', '}', and nested comments will be replaced by '(@*$', '$*@)'.

```
function foo {my comment};
begin
    i := 5;
end; {foo}
```

Would become:

```
{function foo (@*$my comment$*@);
 - begin
 -     i := 5;
 - end; (@*$foo$*@)}
```

If there is no selection, **CommentBlock** treats the current line as the selection.

UnCommentBlock

Removes comment characters that are specific to the current language from the selected text (see also SelectLanguage). The selected text must begin with comment-start symbols and end with comment-stop symbols in order for

UnCommentBlock to work. Otherwise, it will do nothing. If correctly commented, **UnCommentBlock** applies the following rules to the selected text:

- Lines beginning with ’ - ’ will have the ’ - ’ deleted.
- Nested comments detected (i.e. /@*$, $*@/ for C) will be unnested back to their original state.
- The beginning and ending comment markers will be removed from the ends (i.e. for C, /*, and */).

This command should “undo” a **CommentBlock** and is useful for uncommenting old comment blocks in later sessions.

If there is no selection, **CommentBlock** treats the current line as the selection.

LowerCaseBlock

Changes each alphabetic character in the selection to lower case.

UpperCaseBlock

Changes each alphabetic character in the selection to upper case.

JoinLines

Joins the currently selected lines by replacing new lines with spaces. If there is no selection, the current line is joined with the successive line.

Search commands

The MULTI Editor supports two ways to search the open file for text:

- *Interactive search* allows you to perform a variety of searching tasks, such as search-and-replace and regular expression matching, through an interactive dialog box.
- *Incremental search* has fewer options but allows you to see the area of the open file that matches the text you are searching for as you type the search text. This can be useful for quickly finding a piece of text without having to type it completely. Except for the replace facilities in the interactive search dialog box, these commands will not modify the open file.

Search

Opens the interactive Search dialog box. This command is equivalent to Edit > Find. See “Searching” on page 155 for more information.

ISearch

Starts an incremental search that will proceed forward (toward the end of the open file) from the current cursor position. When you run **ISearch**, the word Srch: is displayed in the status bar, indicating that you should begin typing the text that you wish to locate. As you type each character, the next piece of text in the open file that matches what you have typed so far will be selected, and the cursor will be moved to the matching text. If no matching text can be found before the end of the open file, the editor will beep.

If an incremental search is already in progress when this command is executed, the next piece of text after the cursor which matches the characters typed so far will be selected. In this way, each piece of text matching the search text will be selected in succession, once for each time that the **ISearch** command is re-executed. If there are no more matching pieces of text before the end of the open file, the incremental search will "wrap around," proceeding from the top of the open file.

To search the open file using more powerful methods such as regular expression matching, use the **Search** command instead. See also **BackISearch**.

BackISearch

This command is similar to the **ISearch** command, except that it searches backward (toward the beginning of the open file) from the current cursor position. When no more matching text appears before the cursor, the backward incremental search will wrap around, proceeding from the bottom of the open file.

TruncateSearch

Restarts the incremental search at the cursor's current position.

StopSearch

Stops an incremental search. The most recently matched or partially matched text will remain selected, and the cursor will remain at the matched text selection, if any. The **Abort** command will also stop an incremental search.

Undo/Redo commands

These commands allow you to undo, repeat, and cancel other commands and actions that you have executed in the past, or are currently executing.

Undo

Allows you to undo all the changes made to the current file since it was opened. You can undo as many of the changes as you want. For example, suppose you open a file and make three separate edits. Now, you choose undo, then undo again. Your file now contains only the first edit that you made. Typing is merged into a single undo; typing "abc" and then performing an undo will remove all three characters. Equivalent to Edit > Undo.

Redo

Redoes an edit that was undone with the **Undo** command. You can redo any undone edits if you have not yet made any other edits. Equivalent to Edit > Redo.

RepeatLast

Repeats the most recent edit at the cursor's current position. See "To repeat the last change you made to a file" on page 149 for more information.

Abort

Aborts any ongoing command, such as a search.

File commands

These commands allow you to choose new files to edit, as well as save or discard open files.

OpenFile

Format: **OpenFile** [*filename*]

Opens the given file in the current Editor window. If no parameters are specified, then this command opens the Edit File dialog box that allows you to open or create a file. This command is equivalent to File > Open.

LoadFile

Format: **LoadFile** [*filename*]

Opens the given file in either a new editor window or the current editor window, depending on the current setting of the **OpenFilesInNewBuffers** config option. This config option is accessible from Config > Options... > Editor tab > Reuse Editor Windows. If no parameters are specified, then this command opens the Edit file dialog box that allows you to open or create a file.

LoadFileWithNewEditor

Format: **LoadFileWithNewEditor** [*filename*]

Opens the given file in a new Editor window. If no parameters are specified, then this command opens the Edit file dialog box that allows you to open or create a file. This command is equivalent to File > New Editor.

Save

Saves the current file. Equivalent to File > Save.

SaveAs

Opens the Save As dialog box allowing you to enter the name you wish to save the current file. This allows you to save additional copies of the current file under different names. After saving a file under a different name, the current file edited is changed to the new file. Equivalent to File > Save As.

SaveAll

Automatically saves all open files without prompting you with a dialog box.

SaveAllLog

Opens a dialog box that lists all currently edited files that are under version control. You can choose to save any combination of these files by clicking the box next to the file's name. Clicking OK saves all the files selected and allows you to enter the same comment for all of them. This is equivalent to saving all the files at once and entering the same comment for each log. All the files also have exactly the same date and time in their log entry.

QuerySaveAll

Lists all currently edited files modified since the last save. Click the box next to the file's name to select any files to be saved. By default, all modified files are selected. Clicking OK saves all selected files. If any of the files are under version control, it will ask you to enter a comment. The comment is the same for all files saved. Equivalent to File > Save All.

QuerySaveComments

Identical to QuerySaveAll, except if comments are turned off for a file under version control, it will still ask you for a comment.

Revert

Reverts the file to the last saved version (file on disk), deleting any unsaved changes. Equivalent to File > Revert to Saved.

CyclePush

Allows you to edit the previous file in the Editor's stack of open files. Every time you open a file in an existing Editor window, previously opened files are stacked below the newly opened file. Equivalent to the Previous toolbar button.

CyclePushBack

Allows you to edit the next file in the Editor's stack of open files. Every time you open a file in an existing Editor window, previously opened files are stacked below the newly opened file. Equivalent to the Next toolbar button.

EditorFlags

Opens a dialog box to control the file settings for the current file. These include the tag and indent sizes, the time spent on matching parentheses, where text wraps, whether text should wrap, and whether subsequent lines should indent if they wrap. These settings default to the settings in the Config > Options... dialog > Editor tab, and are only applied to the current file in the current session (they are not stored for future sessions). See "Editor tab" on page 254 for more information.

Print

Opens the print dialog box that allows you to print the file or current selection.

Close

Closes the current Editor window. If changes were made to open files, you are prompted to save the files before closing the window. Equivalent to File > Close Editor.

Quit

Opens a dialog box asking which modified files you wish to save, saves the selected files, and quits the entire MULTI session, not just the Editor. Equivalent to File > Exit All.

Done

Performs a QuitAll, except that any files that were not checked out by the Editor during this session are automatically saved without prompting you.

OpenText

This command has been deprecated. Please see the **OpenFile** command.

Tool commands

MULTI provides several useful tools for working with your open files, including searching and diffing files and revisions, and executing shell commands. These commands provide access to those tools.

Grep

Opens a dialog box that asks for text to search for in all of the files in the current program being debugged (if any), as well as any open files. The output from this command is put in a temporary window. Double clicking any of the lines in this window opens an Editor window on the line.

This command works by running the GNU **grep** utility. For your convenience, a copy of GNU grep is installed along with MULTI. However, GNU grep is not part of MULTI and is not distributed under the same license as MULTI. For more information about the GNU General Public License which GNU grep is distributed under, refer to the file gnugrep.README, which is located in the directory where MULTI is installed.

DiffFiles

Opens the Diff Files dialog box that you may use to find and display the differences between two files or between two versions of the same file. This command is equivalent to Tools > Diff Files. See “Comparing files” on page 160 for more information.

MergeFiles

Opens the Merge dialog box that allows you to merge two or three files together. You can also merge versions of the same file together. Also see the Version Control Commands section. This command is equivalent to Tools > Merge Files. See “Merging files” on page 156 for more information.

Minibuffer

Opens a dialog box that you use to execute an Editor command. This command is equivalent to Tools > Execute Editor Commands.

CommandToWindow

Format: **CommandToWindow** [*command* **;**]

Sends the given command to the shell as a command. (The parameter must end with a semicolon ‘;’.) The output is placed into a new Editor window which is given a temporary name. If no parameters are specified for this command, it opens a dialog box that prompts you for a command to send.

ExecuteCmd

Format: **ExecuteCmd** [*command* **;**]

Sends the given command to the shell as a command. (The parameter must end with a semicolon ‘;’.) The command uses the current selection in the editor as stdin, and replaces that selection with stdout. If nothing is selected, the output from the command is inserted into the open file after the cursor’s current position. If no parameters are specified for this command, it opens a dialog box that prompts you for a command to execute.

In the following command, the special macro sequence **%FILE** is replaced by the current file name. Thus, you could implement a PRINT button with the following commands:

NoSelection; ExecuteCmd

lpr %FILE> /dev/null

If there is a current selection, then it sends to the command as standard input. Any output from the command replaces the selection. The following macro sequences are recognized:

%FILE: Replaced with the name of the current open file.

%SEL: Replaced with current selection.

%LINE: Replaced with current line number.

%COMMENTS: Replaced with text from a dialog box that prompts for input.

!

Same as **ExecuteCmd on page 206**.

Shell

Format: **Shell** [*command* **;**]

Sends the given command to the shell as a command. (The parameter must end with a semi-colon ";".) The output is sent to the console from which MULTI was launched. Thus, this command has no effect on the document. If no parameters are specified for this command, it opens a dialog box that prompts you for a command to execute.

Notepad

Opens a small editing window on a scratch file for taking notes. Equivalent to Tools > Notepad..., or the **note** command in the Debugger.

The scratch file used is *~/.Notes*.

cmdprompt2wnd

This command is deprecated. Please see **CommandToWindow on page 206**.

Tag commands

ErrorOrTag

Searches either an output window from the grep command or a progress window for the next item. In grep windows, the next item is the Next tag and file/line number combination. When the next item is found, the insertion point moves to it. A window opens the file the item is from with the appropriate line selected.

The window searched is determined in the following ways:

- If the current window contains output from grep, then it is searched.
- If the current window does not contain output from grep, and the last time this command is used was on a progress window, then the progress window is searched.
- If the current window does not contain output from grep, and the last time this command is used was on a grep window, then the grep window is searched.
- If the current window does not contain output from grep, and this is the first time using this command, then the latest progress window is searched.
- If this command refers to a window that does not exist, for example, if no grep window exists, then MULTI gives you an error.

OpenTag

Enters a new file or procedure to edit in the current window. The new tag is typed at the top of the window, exactly as if clicking the filename. If text is currently selected, then the text is automatically used for the new tag to edit.

For windows produced by the grep command, this opens the file and specifies the line number.

If no text is selected, it is identical to the EditTag command.

NewTag

Searches the tag database for a procedure with the name of the selected text. If the tag is found, the file containing the tag is opened in a new Editor window and the editor attempts to search for the tag.

In windows produced by the grep command, this opens the file and specifies the line number.

SpecialTag

When used inside of a window with output from the progress window, this moves to the location of the next error.

AppendTagFile

Append the entries from the specified tag file into the tag database for later queries.

ResetTags

Clear the current tag database of all entries.

EditTag

This command has been deprecated. Please see the OpenTag and NewTag commands.

Version control commands

CheckIn

Checks the file into version control, prompting you for comments.

CheckOut

Checks the file out of version control.

AllowAutoCheckout

Enables auto checkout from version control. This only affects buffers that are under version control. AutoCheckout means whether typing implicitly checks out a file or whether explicit checkout operations are required. This option is on a per-file basis, and defaults to the value of the “Automatic Checkout” configuration option. See also **PreventAutoCheckout**.

PreventAutoCheckout

Opposite of **AllowAutoCheckout**.

Discard

Discards the current checkout. If the file is checked out, discards the checkout and reverts the contents of the file to those from the last checked in version. This function updates the timestamp on a file in case the modified version was used in a previous build.

PlaceUnderVC

Places the current file under version control, prompting you to insert comments (if desired). If using MVC, this is equivalent to the **create** command, which creates a log file. See “Create log” on page 135 for more information.

vcbuffer

Opens a dialog box that asks you for a version control command. The full name of the current file is appended at the end of the command.

Mvcbuffer

Opens a dialog box that asks you for a MULTI version control command.

RevertDate

Opens a dialog box that asks you to specify the date to revert the file to out of version control (currently only supported for some version control systems (e.g. MVC).

RevertHistory

Opens a dialog box that allows you to select a version to revert the file to out of version control (currently only supported for some version control systems (e.g. MVC). See “ShowHistory” on page 210.

RevertToBackup

Revert the current file to the backup file. Only valid if editor backups are enabled in the configuration options (off by default).

RevertVersion

Opens a dialog box to specify the version number to revert the file to out of version control (currently only supported for some version control systems (e.g. MVC)).

ShowHistory

Opens a dialog box that shows the version-control specific version history. For MVC, this opens a dialog box that shows you the first line of every comment, who checked the file in, and the date. Clicking on an entry opens an editor on a temporary copy of that version.

ShowLastEdit

Note: This command is only supported when using MVC (MULTI Version Control).

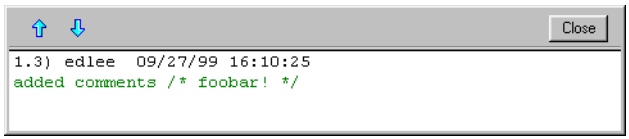
Finds the version of the current file that changed the selected text. This command opens a window on the version that changed the text, placing the cursor at the beginning of the change. Enabled only if the current file is under

version control (such as MVC) that supports this feature. Equivalent to the menu item if you choose Version > Show Last Edit.

The current file must also be checked in. Otherwise, the results of this operation are undefined.

When you run this command, three windows are opened very much like a Diff (Tools > Diff Files... > Diff). The editor window on the left shows the last version which had different text in the selected lines. The editor window on the right shows the next version. Looking at the two versions will show you how the text changed between the two versions.

The third window is the Show Last Edit controller. It consists of a button bar with three buttons and a text section underneath.



Here's a description of the buttons:

- Previous ( ): shows the previous difference (if any) between the two displayed versions.
- Next ( ): shows the next difference (if any) between the two displayed versions.
- 'Close' ( ): closes the controller and both editor windows.

All three windows will also be closed if you close either of the two editor windows.

The text section below the button bar shows information regarding the change between the two versions. This information is laid out as follows:

Version number)	User	Date and Time
Comments		

Where the first line contains the version number, user, and date for the right hand file (the one where the change was checked in). The remaining lines show the comments (potentially several lines) and is displayed in the currently configured comment color.

ShowView

Note: This command is only supported when using ClearCase.

Displays the user's current view.

CreateLog

This command has been deprecated. See **PlaceUnderVC** for more information.

Configuration commands

The MULTI Editor's appearance and behavior can be configured heavily beyond the defaults. These commands allow you to specify values for any of the MULTI Editor's options.

Configure

Format: **Configure** *config_item=value*

Format: **Configure** *config_item***:***value*

Format: **Configure** *config_item value*

This is identical to the Debugger command **configure**. If no parameters are specified, then this command opens a dialog box which requests that you enter a configuration command.

ConfigureFile

Format: **ConfigureFile** [*file*]

This command treats the given file as a list of configuration commands, and executes them in order.

AlterMode

Format: **AlterMode** [*mode number*]

This command is typically only bound to keys, and can be used to create bindings for sequences of key combinations, instead of just single-stroke key combinations.

The actual **AlterMode** command switches the editor to any of 10 modes, called "Edit0" through "Edit9". In each mode, keystrokes have different behavior depending on what their binding is for that particular mode. This is useful because it can cause any given key combination to have up to 10 meanings, depending on whether or not it was preceded by a key that is bound to the **AlterMode** command.

Usually, the **keybind** command only allows you to bind a command to a key with modifiers such as the Control and Shift keys. However, with **AlterMode,** you can bind commands to a key sequence. The **keybind** command requires the specification of an editing mode. When you press the key in that mode, the given command executes. To bind commands to keys to work in the MULTI Editor, the mode should usually be Edit (which refers to the same mode as Edit0). However, if this command is specified and followed by a mode called Edit1-Edit9, then the next key press in the Editor will execute the command that it is bound to the new mode. For example:

```
keybind “x”|Control@Edit=SaveFile
keybind “s”|Control@Edit2=RightD
```

With this command, pressing **Ctrl+x** in the Editor saves the current file. However, pressing **Ctrl+s** in the Editor window does nothing because this command only works in mode Edit2. By default, the Editor mode is Edit or Edit0. However, if you enter the command:

```
keybind “q”|Control@Edit=AlterMode Edit2
```

then pressing Ctrl+q changes the Editor mode to Edit2 for only the next key press. So if you type **Ctrl+q Ctrl+s**, the cursor moves to the right one character, the **RightD** command. **Ctrl+s** works this time because it is now in the correct mode, Edit2.

Any command issued while in an alternate mode will switch the mode back to Edit0, the default mode.

->

This command is only used for specifying menu bindings. It must be followed by the name of a menu, which is created with the specific menu command that opens that menu. Any menus created for the Editor should only contain editing commands.

ShowContextMenu

This command is issued to open a context sensitive menu at the current mouse location. This should be bound only to mouse buttons.

AlterLocation

Format: **AlterLocation** [*mode_number*]

This command is deprecated. Please see the **AlterMode** command.

Help commands

These commands allow you to access the MULTI Editor's help features.

About

Opens the About box which describes version information.

Help

Opens the online help system. This command is equivalent to Help > Editor Help.

Identify

Waits until you enter another command, either by key presses or mouse clicks, and displays the name of that command.

Insert commands

These commands add text to the open file at the cursor's current position.

" " (text surrounded by double-quotes)

Inserts all of the text between double quotes into the open file at the cursor's current position. The current selection, if any, is replaced with the text between the quotes. Standard C quoting sequences (i.e. \n, \t, \\) are allowed.

Tab

Inserts a tab (\t) character into the open file at the cursor's current position. The current selection, if any, is replaced with the tab.

UserName

Inserts the current user name into the open file at the cursor's current position. The inserted name is inserted in lower case.

InsertNewline

Inserts a newline (\n) character into the open file at the cursor's current position. The current selection, if any, is replaced with the newline.

InsertFile

Opens the Insert dialog box, in which you may select a file to be inserted into the current line. The contents of the selected file are placed on the line above the cursor.

EnterInsertMode

Puts the Editor into insert mode, allowing you to enter literal keystrokes into your file, even if the keys are bound to commands.

For example, suppose you customized the Editor so that every time you press d, the cursor moves down one line. This makes it impossible to type the letter d in the file. When you turn on insert mode and press d, the letter d appears in the file, and the cursor does not move down one line.

Insert mode may be turned off by the **FlashCursor** command (see **FlashCursor on page 191**).

Quote

Forces the character generated by the next key press to be literally entered in the file, even if the key is bound to a command. This is useful for entering characters that have commands bound to them.

Beep

Sounds a tone that should be audible to the user.

NextWindow

Moves the pointer into another Editor window and raises that window to the foreground. Using this command repeatedly will eventually cycle through all of the open Editor windows.

ToggleErrorView

Enables error view mode, which is useful for grep and Progress windows.

'if' conditional commands

The conditional command is useful for constructing scripts and key bindings which predicate on a particular state or mode within the editor. For example, **if** commands can be used to make your key bindings more responsive to the existence of selections or whether or not the editor is in insert mode.

if *condition* **{***cmds1***}[else {***cmds2***}];**

This executes the commands given for *cmds1* if *condition* is true, and executes *cmds2* if *condition* is false (although specifying the else clause is optional).

Currently, *condition* may only be one of the following:

<searching>
This is true if an incremental search is currently in progress, and false otherwise. (See the **ISearch on page 201**.)

<noselection>
This is true if there is no primary selection.

<nosselection>
This is true if there is no secondary selection.

<insertmode>
This is true if you are not in insert mode.

<select *num***=line>**
This is true if the selection number *num* aligns on line boundaries.

<select *num***=rect>**
This is true if the selection number *num* is rectangular.

<select *num***=text>**
This is true if neither of the above two are true.

The keyword **else** is included optionally, followed by a second command list. If **else** is included and condition is not true, then the second command list executes.

The final closing brace is followed by a semicolon.

Example

```
if <noselection> {SelectLine}; Cut1
```
If there is no selection, then this selects the entire line. The **Cut1** command is always executed.

```
if <noselection> {ContinueSelection; SOL} else {Delete}
```
If there is no selection, then this selects from the current cursor position to the end of the line. If there is a selection, it is deleted.

Chapter

8

Default key bindings

This chapter contains:

- Default keyboard settings
- Escape key interrupt
- Default mouse settings

This chapter tabulates the default key and mouse combination settings. See Chapter 10, “Configuration commands” for information on how to implement the keybindings and customize your settings.

Default keyboard settings

Moving the cursor

Function	Editor command	Keystrokes
Move the cursor up one line	Up	Up Arrow
		Ctrl+k
Move the cursor up a number of lines (default is 4)	UpSome	Ctrl+Up Arrow
Move the cursor up one page	PageUp	Ctrl+Shift+b
		PageUp
		F29
Move the cursor down one line	Down	Down Arrow
		Ctrl+j
Move the cursor down a number of lines (default is 4)	DownSome	Ctrl+Down Arrow
Move the cursor down one page	PageDown	Ctrl+Shift+n
		PageDown
Move the cursor left one character	LeftU	Left Arrow
		Ctrl+h
Move the cursor right one character	RightD	Right Arrow
		Ctrl+l (lowercase 'L')
Move the cursor to the previous word	ReverseWord	Ctrl+Left Arrow
Move the cursor to the next word	Word	Right Arrow
Move the cursor to the beginning of the line	SOL0	Ctrl+w or Home
Move the cursor to the end of line	EOL	Ctrl+e or End
Move the cursor to the beginning of the next line	EOL; RightD	Ctrl+Enter
Move the cursor to the end of the file	EOF	Ctrl+End
Move the cursor to the beginning of the file	SOF	Ctrl+Home

Selecting text

Function	Editor commands	Keystrokes
Extend selection up one line	ContinueSelection; Up	Shift+Up Arrow
		Ctrl+Shift+k
Extend selection up a number of lines	ContinueSelection; UpSome	Ctrl+Shift+Up Arrow
Extend selection up one page	ContinueSelection; PageUp	Shift+PageUp
		Shift+F29
Extend selection down one line	ContinueSelection; Down	Shift+Down Arrow
		Ctrl+j
Extend selection down a number of lines	ContinueSelection; DownSome	Ctrl+Shift+ Down Arrow
Extend selection down one page	ContinueSelection; PageDown	Ctrl+Shift+ PageDown
		Shift+F35
Extend selection left one character	ContinueSelection; LeftU	Shift+Left Arrow
		Ctrl+Shift+h
Extend selection right one character	ContinueSelection; RightD	Shift+ Right Arrow
		Ctrl+Shift+l
Extend selection to previous word	ContinueSelection; ReverseWord	Ctrl+Shift+ Left Arrow
Extend selection to next word	ContinueSelection; Word	Ctrl+Shift+ Right Arrow
Extend selection to beginning of line	ContinueSelection; SOL0	Shift+Home
		Ctrl+Shift+w
Extend selection to end of line	ContinueSelection; EOL	Shift+End
		Ctrl+Shift+e
Extend selection to beginning of next line	ContinueSelection; Return	Ctrl+Shift+Enter
Select entire document	SelectAll	Ctrl+a

Searching

Function	Editor commands	Keystrokes
Start or continue an incremental search forward	ISearch	Ctrl+f

Function	Editor commands	Keystrokes
Start or continue an incremental search backward	BackISearch	Ctrl+b
Cancel a search	Abort	ESC
Open the search dialog box	Search	Ctrl+Shift+f
		L9
Start an incremental search, then open the search dialog box	BackISearch;Search	Shift+L9

Deleting text

Function	Editor commands	Keystrokes
Delete last character	Backspace	Backspace
Delete next character	if <noselection> { ContinueSelection; RightD }; Delete	Delete
		Ctrl+d
Cut an entire line	SelectToLines; Cut2	Ctrl+m
Merge selected lines	JoinLines	Ctrl+p
Delete previous word	if <noselection> { ContinueSelection; ReverseWord }; Backspace	Ctrl+ Backspace
Delete next word	if<noselection> { ContinueSelection; Word }; Delete	Ctrl+Delete
Cut to beginning of line	if<noselection>{ContinueSelection ; SOL}; Cut2	Ctrl+u

Indenting

Function	Editor commands	Keystrokes
Indent a line	Indent	Ctrl+i
Unindent a line	Unindent	Ctrl+Shift+i
Auto-indent the selected line(s)	if<beforenonwhite> {AutoIndentOrTab} else {if <noselection> {Tab} else {AutoIndentOrTab} }	Tab

Copying, cutting and pasting

Function	Key	Modifiers	Command
Copy to clipboard	L6	n/a	Copy1
	c	C	Copy1
Copy to second buffer	L6	S	Copy2
	C	C,S	Copy2
Copy to third buffer	L6	C	Copy3
	c	M, C	Copy3
Copy to fourth buffer	L6	C, S	Copy4
	C	M, C, S	Copy4

Function	Key	Modifiers	Command
Cut to clipboard	L10	n/a	Cut1
	x	C	Cut1
Cut to second buffer	L10	S	Cut2
	x	C, S	Cut2
Cut to third buffer	L10	C	Cut3
	x	M, C	Cut3
Cut to fourth buffer	L10	C, S	Cut4
	x	M, C,S	Cut4
Paste from clipboard	L8	n/a	if <noselection> { if <select1=lines> {SOL0}} Paste1
	v	C	if <noselection> { if <select1=lines> {SOL0}} Paste1
Paste from second buffer	L8	S	if <noselection> { if <select1=lines> {SOL0}} Paste2
	v	C, S	if <noselection> { if <select1=lines> {SOL0}} Paste2
Paste from third buffer	L8	C	if <noselection> { if <select1=lines> {SOL0}} Paste3
	v	M,C	if <noselection> { if <select1=lines> {SOL0}} Paste3
Paste from fourth buffer	L8	C, S	if <noselection> { if <select1=lines> {SOL0}} Paste4
	v	M, C, S	if <noselection> { if <select1=lines> {SOL0}} Paste4

Fixing errors

Function	Key	Modifiers	Command
Undo last edit	L4	n/a	Undo
	z	C	Undo
Redo last undo	L2	n/a	Redo
	y	C	Redo
Load version that changed text	r	M, C, S	ShowLastEdit

File commands

Function	Editor commands	Keystrokes
Close the current editing window	Close	Ctrl+q
Save the current file	Save	Ctrl+s
Save the current file under a new name	SaveAs	Ctrl+Shift+s
Edit the tag the cursor is near in a new window	SelectWord; NewTag	Ctrl+t
Open a dialog box to open a file in the current window	OpenFile	Ctrl+o L7
Open a dialog box to open a file in a new window	LoadFile	Ctrl+n Ctrl+Shift+o
Enter a new line number to go to	EditLine	Ctrl+g
Cycle through open file buffers forward	CyclePush	Ctrl+Tab
Cycle through open file buffers backward	CyclePushBack	Ctrl+Shift+Tab
Save current file, then close window	Done	Ctrl+Shift+q

Debugging

Function	Command	Keystrokes
View the procedure one higher on the call stack	UpStack (E +)	Ctrl++ (plus)
View the procedure one lower on the call stack	DownStack (E -)	Ctrl+- (minus)
Help	Help (help)	F1
Run a program if not started or continue executing if a program has stopped	Go (C)	F5

Function	Command	Keystrokes
Continue to the end of the current subroutine and stop in the calling routine after returning	Return (cU)	F9
Execute single statements and step over procedure calls	Next (n)	F10
Execute single statements and step into procedure calls	Step (s)	F11 (May not work on some keyboards.)

Miscellaneous

Function	Editor commands	Keystrokes
Flash the line the cursor is on	NoSelection; FlashCursor	ESC
Enter the next key press sequence literally	Quote	Ctrl+\ (backslash)
Insert a new line	InsertNewline;AutoIndent	ENTER
Insert a new line at the end of the current line	EOL; InsertNewLine	Ctrl+Shift+r
Repeat the last command	RepeatLast	Ctrl+. (period)
Transpose previous two characters	NoSelection; ContinueSelection; Left; Cut2; Left; Paste2; Right	Ctrl+Shift+t

Escape key interrupt

Pressing ESC during a while, step or blocking run will cause MULTI to abort the command and clean up. In the case of a step or blocking run, the process will be halted and the PC displayed.

Default mouse settings

Key for Mouse Clicks:

LC = Single left click

LLC = Double left click, LLLC = Triple left click, etc.

MC = Single middle click

MMC = Double middle click, MMMC = Triple middle click, etc.

RC = Single right click

RRC = Double right click, RRRC = Triple right click, etc.

Ctrl = Control Key

First (leftmost) mouse button

Function	Mouse operations
Start new (primary) selection	LC
Select text for the primary selection	LC+drag
Extend your selection to the mouse pointer	Shift+LC
Select the current word	LLC
Select the current line	LLLC
Select the entire file	LLLLC
Select text and copy it to the clipboard	Ctrl+LC
Extend your selection to the mouse pointer and copy it to the clipboard	Ctrl+Shift+LC
Select the current word and copy it to the clipboard	Ctrl+LLC
Select the current line and copy it to the clipboard	Ctrl+LLLC
Select the entire file and copy it to the clipboard	Ctrl+LLLLC
Select text and cut to clipboard	Meta+LC+drag
Select the current word and cut to clipboard	Meta+LLC
Select the current line and cut to clipboard	Meta+LLLC

Second (middle) mouse button

The middle button may not be available, depending on your mouse.

Function	Mouse operations
Make a secondary selection and either delete it or replace it with a primary selection	MC
Replace a current word with primary selection	MMC
Replace a current line with primary selection	MMMC
Replace an entire file with primary selection	MMMMC
Make a secondary selection and replace with clipboard	Ctrl+MC
Replace a current word with clipboard	Ctrl+MMC
Replace a current line with the clipboard	Ctrl+MMMC

Function	Mouse operations
Makes secondary selection and replace with clipboard	Meta+MC
Replaces current word with clipboard	Meta+MMC
Replaces current line with clipboard	Meta+MMMC

Third (right-most) mouse button

Function	Mouse operations
Open the pop-up menu for performing operations on the object you just clicked	RC
Edit the current tag in current window	Meta+RC
Edit the current tag in a new window	Meta+RRC
Select a matching parenthesis	Shift+RC
Select from the current location to the first matching parenthesis	Shift+RRC
Open a window for taking notes	Ctrl+Shift+RC
Select word, open a tag	Meta+RC
Select word, open a tag in new window	Meta+RRC

Chapter

9

Configuring and customizing MULTI

This chapter contains:

- Setting configuration options
- Customizing the graphical user interface (GUI)
- Creating custom functionality
- How MULTI uses startup files to configure a session
- Example customizations

MULTI gives you the ability to configure and customize your Integrated Development Environment (IDE) to fit the way you work most efficiently. You can:

- Set configuration options
- Customize the MULTI graphical user interface (GUI)
- Create custom functionality

Setting configuration options

MULTI provides standard options governing MULTI's appearance and behavior that you can edit according to your preferences. You can save these options in a configuration file (*.cfg), then have MULTI reconfigure itself at startup based on existing configuration files. Alternatively, you can manually load a configuration file during a MULTI session to reconfigure your environment. For more information about specific configuration settings, see Chapter 10, "Configuration commands".

Editing configuration options

You can edit MULTI's configuration options in two ways: using the Config menu and using the **configure** command. Both methods write your settings to the same temporary configuration file; when you edit a setting using the **configure** command, your changes are automatically reflected in the Config menu dialog boxes.

Be aware that changes that you make are NOT automatically saved for future MULTI sessions. You must manually save your configurations if you want them restored the next time you run MULTI.

Config menu

To make changes to MULTI's options using the graphical configuration interface, choose Config > Options....

configure command

If you know the name of the options you want to change, you can use the **configure** command from the Debugger Command Pane. The format is:

configure *config_item*=*value*
configure *config_item*:*value*
configure *config_item value*
configure ?

Typing **configure ?** displays a list of all items you can configure. You can change individual items by including their name followed by an equal sign (‘=’), a colon (‘:’), or a space (‘ ’), then the new value. For example:

```
configure tabsize=9
configure background #ffffff
configure moon: On
configure linenumbermode: Both Numbers
configure prompt: "MULTI> "
configure status.stopped STOPPED
configure key: "Up"@Command=backhistory
```

Saving configuration options for future MULTI sessions

You must save your option settings if you want to preserve them for future MULTI sessions. When you save the settings, you are saving a *.cfg file which can be loaded into MULTI in future session.

- If you want to save the current settings so that they are loaded every time you start MULTI, choose Config > Save Configuration As Default. When a user chooses this menu item, the current options are saved in the user configuration file, which loads each time the same user starts MULTI. For more information about the user configuration file, see “user configuration file” on page 238.
- If you want to save the current settings, but don’t want them to be loaded automatically every time you start MULTI, choose Config > Save Configuration, then save the *.cfg file.
- If you want to save the current settings so that they are always loaded for every user in a user group, choose Config > Save Configuration, then save the file as: $MULTI’S_INSTALLATION_DIR/config/**multi.cfg**. For more information, see “global configuration file” on page 237.

You can also manually create and edit a *.cfg file.

Loading configuration files

When MULTI starts a session, it automatically configures itself based on the following configuration files (*.cfg), if they exist:

- global configuration file
- user configuration file
- a configuration file specified on the commands line with the **-config** option

Be aware that settings in a configuration file can override global or user options, and that MULTI also loads script files that can affect configuration options. For more information about what files MULTI uses at startup, see "How MULTI uses startup files to configure a session" on page 237.

Loading a configuration file during a session

To load a predefined configuration file during a MULTI session, do one of the following:

- Choose Config > Load Configuration, then select the configuration file you want to load.
- In the Debugger Command Pane, use the following command:

  configurefile *filename*

Configuration file format

Configuration files consist of lines, each one of which can be blank, a comment, or a configuration. Blank lines contain only whitespace characters and are ignored. Comment lines start with a pound sign (#) and are ignored. For example:

```
tabsize: 8
background: #ffffff
moon: On
linenumbermode: Both Numbers
prompt: "MULTI> "
status.stopped: STOPPED
key: "Up"@Command=backhistory
```

Config menu

The following is an overview of the menu items in the Config menu.

Options...

Opens a tabbed dialog that allows you to set most of the configuration options that affect visual and behavioral aspects of MULTI.

Save Configuration as Default

Saves the current configuration of MULTI into the user configuration file, which MULTI reads each time it starts to restore your configuration to the state it was in when you saved it. MULTI doesn't ask if you want to save changes to your configuration when you quit, so you have to remember to do so.

See “user configuration file” on page 238 for the location of the user configuration file and other files which MULTI reads automatically on startup.

See “Configuration file format” on page 232 for the format of configuration files (.cfg files).

Clear Default Configuration...

Deletes the user configuration file.

Save Configuration...

Like Save Configuration as Default, but lets you choose a file to save the configuration into. This saved configuration will not be automatically loaded at start up. This can be useful in conjunction with **configurefile** or Load Configuration.... (see “Loading a configuration file during a session” on page 232). For the format of configuration files (.cfg files), see “Configuration file format” on page 232.

Load Configuration...

Loads a configuration file of your choosing.

Customizing the graphical user interface (GUI)

MULTI allows you to customize buttons, menus, keystroke combinations, and mouse clicks in the Debugger, Builder, and Editor. You can remove these GUI elements, add new GUI elements, or change the commands that are executed when the GUI element is used. You can also define a button, keystroke combinations, or mouse click to display a custom menu (see “Opening menus” on page 275).

You can define customized GUI elements in two ways:

- Use the Config > Options dialog box to edit a configuration file (*.cfg). For more information about saving and loading configuration files, see “Setting configuration options” on page 230. If you customize using the Config > Options dialog box, you may need to refer to the corresponding commands in Chapter 10, “Configuration commands”for proper syntax; you can use the table below to determine the appropriate command.
- Write a script file that contains customization commands. The table below lists the commands you need to execute for each GUI element. Be aware that the script files that MULTI loads automatically (global, user, and program script files) are loaded only when the Debugger is active, so don’t

put customizations that apply to the Builder and Editor into these startup script files; use a configuration file instead. For more information about writing a script, see “Scripting” on page 234.

Once you have defined a configuration file or script file that customizes the GUI, you can have MULTI load that file. For more information about these and other files that MULTI uses at startup, see “How MULTI uses startup files to configure a session” on page 237.

Use the following table as a guide to where you need to go to customize a particular GUI element.

To customize	using a configuration file, choose	using commands, see
Debugger buttons	Config > Options > Debugger tab > Configure Debugger Buttons	**clearbuttons on page 266** **debugbutton on page 266**
Editor buttons	Config > Options > Editor tab > Configure Editor Buttons	**editbutton on page 268** **cleareditbuttons on page 268**
menus	Config > Options > General tab > Menus...	**menu on page 273** **clearmenus on page 273**
keystroke combinations	Config > Options > General tab > Key Bindings...	**keybind on page 269** **clearkeys on page 269**
mouse clicks	Config > Options > General tab > Mouse Bindings...	**mouse on page 275** **clearmice on page 275**

Creating custom functionality

You can customize MULTI to perform complex procedures that automate and simplify tasks, configure MULTI automatically based on specific situations, and perform tests, including regression tests. You create custom functionality using scripts and macros.

Scripting

A script is a list of commands in a file that MULTI reads and executes as if they were entered individually in the Debugger Command Pane. They typically end with **.rc**.

A script can contain both commands and expressions. It can contain **if () {} else {}** statements that could, for example, compare a program variable to a particular value and then perform some action based on that result.

You can use scripts for automating common tasks and for regression testing. By writing a command script that executes parts of your program and checks that

your program is running correctly, you can rerun this script at a later date, after making supposedly unrelated changes, to verify that your program still runs the way you expect. See “Example 1: Connecting to a target from MULTI” on page 239, for an example.

Creating a script

You can create a script automatically by use the > *file* command (see “Record and playback commands” in Debugging with MULTI 2000).

You can also manually create a script by putting the commands and expressions into a text file.

Running a script

To run a script, do one of the following:

- Use the < *file* command (see “Record and playback commands” in Debugging with MULTI 2000).
- Specify the script file on the command line (see “-rc file” on page 8).
- Customize a button or menu item to execute the script. Define < *file* as the command that is executed by the button or menu item.

In addition, you can save a script as one of the following startup scripts, which MULTI runs automatically at specific times:

- global script file (see “global script file” on page 238).
- user script file (see “user script file” on page 238).
- program script file (see “program script file” on page 238).

Checking the syntax of your script

Syntax checking checks the validity of your command syntax without causing target interactions or changing system settings. You can check your script’s syntax using:

- The **sc** command (see Debugger Commands in the *MULTI Debugger Reference Manual*) checks a script to make sure it has correct syntax and valid object references (references to variables, etc.).
- The **bpsyntaxchecking** configuration option controls whether MULTI checks the syntax of the commands associated with breakpoints when the breakpoint is set.

In many debugging environments, it would be tedious to repeatedly run into syntax errors minutes or hours into a lengthy auto-debugging process. You

could manually find the syntax errors by starting the process, making sure that all the breakpoints are hit and the associated commands are executed, and testing all of the script branches, but that can be very time-consuming. A better option is to use the **sc** command to check the syntax of your scripts before putting them into service.

The **sc** command has three limitations:

1. It can't check syntax errors in commands associated with a breakpoint set with the **bu** command. The **bu** command sets up-level breakpoints; the context of the breakpoint depends on the dynamic execution. For example, in the command **bu {print varA}**, MULTI cannot determine the up-level procedure until the **bu** command is actually encountered while running the script. So, it has no way to check if the variable **varA** is a valid reference when syntax checking the script.
2. **sc** cannot check the syntax errors in the body of a MULTI macro.
3. **sc** treats all local variable references that are not in a breakpoint command as errors, as in the following script:

```
b main#10 {if (argc>2) {print argc+i;} else {print
"Too few arguments"}}
print argc+i;
print global_var;
```

If the procedure **main** contains the number variables **argc** and **i**, and **global_var** is a global variable, **sc** will pass the first and the third lines. But **sc** treats the second line as error because MULTI cannot determine the context in which the **print argc+i;** command will be performed.

Macros

Macros are available through the **define** command. The **define** command is similar to the C preprocessor directive **#define**. It gives you the ability to create a macro inside MULTI. You can then later run that macro from the Debugger command pane or a script. See "Example 2: Regression testing" on page 239 for an example.

Macros can combine a small set of commands into one macro function, just as **#define** macros are used in C programs. A macro can return a value and be used in an expression evaluated in the Debugger Command Pane, just as a **#define** macro. Although you can probably automate most tasks by using commands and scripting, you might want to create macros in some circumstances.

How MULTI uses startup files to configure a session

Certain configuration files are executed automatically to set up the environment whenever MULTI starts up, and additional script files are executed whenever the Debugger starts on a particular program. MULTI parses these files in the following order (if a file does not exist, it is skipped):

1. global configuration file
2. user configuration file
3. command line configuration file
4. global script file (Debugger environment only)
5. user script file (Debugger environment only)
6. command line script file (Debugger environment only)
7. program script file (Debugger environment only)

Because changes to the same configuration item (see Chapter 10, "Configuration commands") can exist in multiple files, the execution order of such files is important. A configuration change can override the effects of a previous change.

The global config file, user config file, and command line config file are parsed on MULTI startup, whether the Builder comes up first or the Debugger. The global script file, user script file, and command line script file are executed once when MULTI's first Debugger window appears (the first time you start a Debugger on a program from an invocation of MULTI). The program script file is executed every time that program is loaded into a Debugger window, i.e., on every new Debugger window on that program, on every program reload (debug command with no arguments), but not on every program restart.

If you load a new program into an existing Debugger, MULTI will automatically execute the commands in the new program's script file (if any), but does not clean up the effects of the old program's script file (if any).

All Debugger windows launched in a debugging session share the same Debugger environment. If you launch multiple Debugger windows from the same MULTI program (for example, the Builder), all the script files of the corresponding debugged programs will be executed, resulting in the final Debugger environment.

global configuration file

$MULTI'S_INSTALLATION_DIR/config/**multi.cfg**

This file is useful in setting up a common environment, required by whole user groups when debugging any program. If you run the MULTI editor as a stand-alone executable (not from within MULTI(#) it uses me.cfg instead of multi.cfg.

user configuration file

$HOME/.ghs/multi.cfg

This file is useful in setting up a common environment, required by a single user when debugging any program. If you run the MULTI editor as a standalone executable (as opposed to from within MULTI), it looks for me.cfg instead of multi.cfg.

command line configuration file

You can specify a configuration file on the MULTI command line with **-config filename** or **-configure filename**. This feature is not available when running the MULTI editor as a separate executable.

global script file

$MULTI'S_INSTALLATION_DIR/config/**multi.rc**

This file is useful for commands which need to run once when any person in a whole user group starts the Debugger the first time in a debug session.

user script file

$HOME/.ghs/multi.rc

This file is useful for commands which need to run once when debugging any program by a single user.

command line script file

You can specify a script file in the command to run MULTI with the **-rc** option, such as **multi my_prog -rc your_script**.

program script file

$EXECUTABLE_DIR/executable_name.rc

Example customizations

You can customize MULTI to help you work more efficiently. For example, you can create a script file that defines a new button in the MULTI Debugger which executes a script that contains a list of Debugger commands and may also define and make use of a macro. You could also define a Debugger button from a configuration file. You now have a Debugger button which can perform a complex task. The following two examples show some automating possibilities.

Example 1: Connecting to a target from MULTI

You can automate the series of repeated steps required to connect to a target. With the large number of connection methods and the variability of specific user environments, automating a connection is generally unique to a user. MULTI lets you define exactly what you want automated.

In this example, the target processor is a PPC860, the target board is Motorola PPC860ADS, and MULTI is connected to it via an HP Processor Probe. The program being debugged is **foo**. The following scripts give you a button in MULTI that will connect MULTI to the target and initialize it.

File: **foo.rc**

```
debugbutton “My Connect” c="< connect.txt" i=connect
```

File: **connect.txt**

```
remote hpserv 192.67.44.234
target rst
target ads setup
```

Example 2: Regression testing

This regression test consists of a program with a function that calculates Celsius from a given Fahrenheit.

File: **foo.c**

```
#define CONV (5.0/9.0)
extern int mytotal;
int celsius (int fahrenheit) {
   int rval = (int) ((fahrenheit - 32) * CONV);
   return rval;
}
void main (void) {
   int some_degrees;
```

```
   int some_celsius;
   some_celsius = celsius(some_degrees);
}
```

File: **foo.rc**

```
debugbutton RegTest1 c="<bar.txt" i="letter_a"
define check_celsius(arg) {
   if (some_celsius != arg) {
      print "Failed!"
      printf ("Failed!\n actual:%d\n expected:%d\n",
              some_celsius, arg);
   } else {
      print "Pass";
      printf ("Pass!\n actual:%d\n expected:%d\n",
              some_celsius, arg);
   }
}
```

File: **bar.txt**

```
b main
r
some_degrees = 45;
S
check_celsius(7);
```

Now, when you start up a MULTI Debugger on **foo**, MULTI runs the script **foo.rc** automatically. **foo.rc** creates a button (with name **RegTest1** and built-in icon **letter_a** with the shape of 'A') and also defines a macro. You can run the regression test by clicking the button. Notice that:

- commands such as **b** are used as if you entered them directly;
- program variables and **#define** macros are referenced;
- functions that are linked into your program like **printf()** are callable;
- C-like expressions such as **if() {}then{}** are evaluated.

This file exists in the same directory as the executable and shares the same name as the executable, but with an **.rc** extension. This file is useful for commands that need to execute whenever the Debugger starts on a particular executable.

Chapter

10

Configuration commands

This chapter contains:

- Options dialog box
- Other Configuration options

The previous chapter discussed how you can customize your interface using MULTI's configuration options. This chapter describes each configuration option and it includes the default settings, if applicable. The chapter begins with options you can access from the Config Options window, as well as the command line or a config file. It is followed by options which can only be set from the command line or a config file.

The format of each entry in this chapter is:

Name in GUI

Format: (command line name)

Default: (if any)

Explanation of option. In cases where the sense of the config option is reversed depending on whether it is accessed through the GUI, through a config file, or the command pane, the default is for the option as it is set in config files and in the command pane, not in the GUI. For config options where there are choices in a pull-down menu, the command line choice may differ from the choice presented in the GUI. For this situation, the GUI choice will be listed followed by the command line choice in parentheses. For example: Use Color Offsets (Offset).

Note: Remember to save your configuration using Config > Save Configuration as Default before you quit MULTI, otherwise you will lose your changes.

Options dialog box

This section describes the Options dialog box (when you choose Config > Options...).

General tab

(Config > Options... > General tab)

Save window positions and sizes

Format: **rememberwindowpositions**

Default: On

Remembers the position and size of each type of window so that the next time a window of the same type is created, it will be created with the same size and position. The first window of a given type is positioned in the saved location, but subsequent windows of the same type are positioned slightly offset from the

previous window (although they are still sized to the same size). This takes effect even across sessions (after you exit and restart the program). For example, if you resize the Builder window, move it to a specific location on your screen, and then close the window, the next time a Builder window is created it will appear in the same exact place. Not all windows are affected by this option. In particular, data explorer windows are not affected.

Use icons for buttons

Format: **iconifiedbuttons**

Default: On

Uses icon based buttons instead of textual buttons.

Display close (x) buttons

Format: **closebuttonontitlebar**

Default: On

Check to have MULTI always provide a close button of its own on windows it creates. This option will only affect new windows created after you change the setting, except for Debugger windows which will adjust to the new setting immediately.

Match exact case in searches

Format: **exactcase**

Default: Off

Determines the case sensitivity of all text searches in MULTI. This option can also be set on a per-editor-window basis with the Edit > Find... dialog.

Allow beeping

Format: **beep**

Default: On

Determines whether MULTI beeps. Enabled, MULTI beeps on various error conditions, such as a search that doesn’t match anything. Disabled, MULTI never beeps.

Show tooltips

Format: **tooltips**

Default: On

To enable tooltips, turn this option on. Tooltips are the little explanatory boxes that pop up when you hover the mouse cursor over something.

Warp pointer

Format: **warppointer**

Default: Into Dialog

Controls when MULTI warps the mouse pointer across the screen to a convenient location.

Warp pointer values		
Value	**Command**	**Meaning**
Never	never	Never move the mouse pointer; the user has complete control over its location.
Into Dialog	IntoDialogue	Warp the mouse pointer onto the default button of new dialog boxes that appear.
In & Out of Dialog	In&OutDialogue	Warp the mouse pointer onto the default button of new dialog boxes that appear, and then back to where it was when the dialog box is closed.

Print command

Format: **printcommand**

Default: lpr

The UNIX command to print a postscript file. This command will be used whenever MULTI tries to print anything.

Vertical scroll bar location

Format: **scrolllocation**

Default: Right

Location of the vertical scroll bar (Left or Right).

Horizontal scroll bar location

Format: **scrollhlocation**

Default: Bottom

Location of the horizontal scroll bar (Top or Bottom).

Display moon phase

Format: **moon**

Default: Off

Display the approximate phase of the moon in the nook of the vertical and horizontal scroll bars instead of the Green Hills Software logo.

Scroll bar width

Format: **scrollbarwidth**

Default: 18

Width of scroll bars.

Main Font...

Format: **font**

Default: "-dt-interface user-medium-r-normal-m*-*-*-*-*-*-*-*-*"

Change the main font, used for code and filenames in the Builder. This font should normally be set to a fixed width font, so that text will line up properly.

Button Font...

Format: **buttonfont**

Default: "-*-interface system-medium-r-*-*-12-*-*-*-*-*-*-*"

Change the button font, used on buttons, menus, and other GUI controls.

Kanji Font...

Format: **kanjifont**

Default: (none)

Change the 16-bit font used to display Kanji text.

Menus...

Opens a dialog box which allows editing of the menus in MULTI using the same format that the menu command takes. See also **menu on page 273**.

Mouse Bindings...

Openss a dialog box which allows editing of the mouse bindings using the same format that the mouse command takes. See also **mouse on page 275**.

Key Bindings...

Opens a dialog box which allows editing of the key bindings using the same format that the keybind command takes. See also **keybind on page 269**.

Online Help...

Opens a dialog box which contains additional configuration options for the online help system. See Online Help Options.

Online Help Options

(Config > Options... > General tab > Online Help)

Help browser

Format: **helpbrowser**

Default: netscape

Web browser to use for online help. Can also be used to give command line arguments to the browser. For example, to set the display to a different machine:

"/home/site/bin/netscape -display othermachine:0.0"

If the browser isn't in your path, the full path must be provided.

Use current context to resolve help ambiguities

Format: **helpcontextdisambiguates**

Default: On

Enabled, the online help system will take the current context into account if you ask for help on a help keyword for which there are multiple help entries. Disabled, the online help system will always ask you to pick which context you are interested in when there is ambiguity.

Browser supports -remote command line option (Netscape)

Format: **helpnoremotecommand**

Default: Off (On in GUI)

NOTE: The meaning of this option varies depending on whether you set it from the GUI, or from the command line or a config file.

If enabled in the GUI, the debugger will try to use Netscape's -remote facility so that the same browser can be used for multiple help sessions. If disabled in the GUI, then help will always open a new web browser each time you request help.

Help in new browser window

Format: **helpinnewwindow**

Default: On

To launch help in a new Netscape window when the browser is already open, turn this option on. Not applicable when helpnoremotecommand is on.

Use Java (1.1) applet for online help

Format: **helpnojava**

Default: On

NOTE: This meaning of this option varies depending on whether it is accessed from the GUI, or using the command line or a config file.

To use Java-based help (with features such as full-text searching), turn this option on. With this option off, a set of html-only pages will be used. Java-based help requires a browser that supports Java 1.1 or greater.

Help port number

Format: **helpportnumber**

Default: 5150

Sets the port number that help will try to bind to. Must be between 1024 and 2^16-1. No effect if helpnojava is on.

Number of ports to scan if bind fails

Format: **helpnumportstoscan**

Default: 100

If the connection is unsuccessful when connecting on the port specified by helpportnumber, this many ports after that one are scanned to find a port to bind to.

Debugger tab

(Config > Options... > Debugger tab)

Ask before halting to set breakpoint

Format: **verifyhalt**

Default: On

When enabled, the Debugger will ask before halting the process to set a breakpoint. Disabled, the Debugger will automatically halt, set the breakpoint, and continue the process without requiring user intervention.

Use procedure relative line number (vs. file relative)

Format: **procrelativelines**

Default: On

Make Debugger commands such as the **e** command interpret line numbers as procedure relative instead of file relative by default. You can obtain the non-default behavior by using the '#' character in the invocation of **e**.

Display all numbers/characters as hex

Format: **hexmode**

Default: Off

Display all numeric values as hexadecimal. Disabled, the display format is chosen based on the "natural" display format for that type. For integral types, the "natural" display format is decimal.

View unsigned char as integer

Format: **viewunsignedcharasint**

Default: Off

Make the "natural" display format of unsigned chars the same as the natural format for ints. This is useful when you want to view byte values as numerical values instead of characters. Disabled, the natural display format for unsigned chars is a literal character, such as 'A'.

Remember breakpoints

Format: **rememberbreakpoints**

Default: Within Session

Determines whether the Debugger should remember breakpoints that you have set for a program the next time you debug the same program.

Never - the Debugger will clear all breakpoints whenever you load or reload a program.

Within Session (WithinSession) - the Debugger will remember breakpoints that you have set for a program when you reload the program during a single session (i.e. until you exit MULTI).

Across Sessions (AcrossSessions) - the Debugger will remember breakpoints for a program even if you exit and restart MULTI.

Coloring for multiple debuggers

Format: **backgroundmode**

Default: Offset from current

Controls the background color of debuggers other than the first one. It is useful to turn this on when using multiple debugger windows if it helps you keep track of which is which.

Off - All debugger windows use the normal background color (see).

Use Color Offsets (Offset) - The subsequent debuggers use predetermined offsets from the normal background color. This option is usually the best since it will pick colors near the current background color, and will keep the text as legible as possible.

Preset high contrast (Preset) - The subsequent debuggers use a set of pre-chosen colors.

Line numbers in source pane

Format: **linenumbermode**

Default: Both Numbers

Controls which line number(s) are displayed on the left side of the Debugger Source Pane.

No Number (None) - No line numbers are displayed.

File Number (File) - Line numbers in file are displayed.

Proc Number (Proc) - Line numbers in procedure are displayed.

Both Numbers (Both) - Line numbers in both procedures and in files are displayed.

Position of buttons

Format: **debugbuttonsloc**

Default: Top

Controls the position of the Debugger buttons

Top - Below the menu bar and above the source pane.

Bottom - Below the command pane and above the status bar.

Command pane height in lines

Format: **cwindlines**

Default: 10

Number of lines in the Command Pane.

Command pane prompt

Format: **prompt**

Default: “MULTI> ”

Text which acts as a prompt just before any commands you enter in the Debugger Command Pane.

Configure Debugger Buttons...

Opens a dialog which allows editing of the debugger buttons using the same format that the debugbutton command takes. In the dialog that comes up, there is a list of buttons on the left and a list of available icons on the right.

More Debugger Options...

Opens the More Debugger Options... dialog which contains additional configuration options for the debugger.

Data Explorer Options

Minimum initial size (WxH)

Format: **minviewsize**

Default: 40x3

Minimum initial size of a data explorer window (also known as view window). If left blank, MULTI will auto-size the data explorer windows appropriately.

Maximum initial size (WxH)

Format: **maxviewsize**

Default: 40x42

Maximum initial size of a data explorer window. If the minviewsize (see above) is greater in either dimension than the maxviewsize, the maxviewsize wins.

Initial position (XxY)

Format: **firstposition**

Default: 0x0

Initial position from the top-left of the screen for a data explorer window, specified in characters and lines. To give the coordinates in pixels, put a 'p' after them (e.g. "100x100p"). This only applies to the first data explorer window; subsequent ones are offset some, avoiding excessive overlap. If left unspecified (blank), the debugger auto-positions all the data explorer windows.

Two color mode

Format: blackwhite

Default: Off

Use only the foreground and background color (except in icons). This increases usability in low-graphics-bandwidth situations, such as when displaying MULTI on a different machine.

Load Color Scheme...

Opens a dialog which lets you choose from a list of prepared color schemes. If you prefer a fundamentally different look from the default, you can try these to find the one that's closest to what you want, then modify the colors.

More Debugger Options...

(Config > Options... > Debugger > More Debugger Options...)

Automatically dereference pointers

Format: **derefpointer**

Default: On

To have MULTI automatically follow pointers when it encounters them instead of just printing the pointer value, enable this option. When on, MULTI shows the value of the pointer and what it points to. Disabled, MULTI only shows the value of the pointer.

Check syntax of breakpoints when they are set

Format: **bpsyntaxchecking**

Default: On

Ensure that the commands being associated with a breakpoint pass syntax checking when the breakpoint is set. With this option on, breakpoints whose commands fail syntax checking cannot be set. Disabled, a syntax error in a

breakpoint command will not be detected by MULTI until the breakpoint is hit and MULTI tries to execute the breakpoint's commands.

Continue running script files on error

Format: **continueplaybackfileonerror**

Default: Off

Ignore errors in script files (.rc files). Disabled, the Debugger will stop running a script file if an error is encountered.

"s" (step) and "n" (next) are blocking by default

Format: **blockstep**

Default: Off

Ensure that no Debugger commands execute until a step or next finishes. Disabled, scripts that use **s** or **n** can appear to behave inconsistently since subsequent commands can appear to be lost sometimes, when in reality they just happened before the step or next finished. A step or next can be made blocking or non-blocking regardless of the setting of this option by appending an **n** (for non-blocking) or **b** (for blocking) to the command.

Show locations of variables

Format: **showaddress**

Default: On

Print the location of a variable (address in memory or register name) before printing the value of the variable when using the **print** command.

Display typedef type instead of basic type

Format: **leavetypedef**

Default: Off

Display the typedef'ed name of a type instead of the actual (self-contained) type.

Show position in non-GUI (-nodisplay) mode

Format: **showposinnodisplaymode**

Default: On

Print the current line after every command in non-GUI mode.

Repeat last command on return key in non-GUI (-nodisplay) mode

Format: **disablecarriagereturnrepeat**

Default: On (Off in GUI)

The meaning of this command varies according to whether you access it through the Config menu, through the command line, or through a config file.

To make the return key repeat the last entered command when running in non-GUI mode, turn this option on in the GUI.

If setting this option from the command line or a config file, turn this option on to *disable* repetition of the last command via the return key.

Stepping over C++ exception or longjmp

Format: **longjmpstepmode**

Default: Ignore/RunAway

When nexting over a subroutine that calls longjmp or does a C++ exception throw (a call to longjmp internally), the subroutine never returns in the normal way. This is significant because the debugger uses a temporary breakpoint just after the normal return to effect the next (this is also true of step if there is no source available for the subroutine). When longjmp is called, this temporary breakpoint is bypassed and execution can “run away” instead of just nexting.

This configuration option controls how MULTI handles this situation.

Ignore/RunAway (IgnoreRunAway) - Let the code call longjmp without worrying about the consequences to a next.

Minimize Temp Stops (MinimizeTempStops) - Fix the problem in a way that doesn’t cause temporary stops in longjmp as the program runs normally under the debugger. This option inserts and removes a temporary breakpoint for each next over a subroutine.

Maximize Step Speed (MaximizeStepSpeed) - Fix the problem in a way that minimizes the time it takes to do a next. This option leaves a breakpoint in longjmp, and will result in a temporary stop if longjmp is called when the program is running normally under the debugger.

Command pane buffer size in bytes

Format: **ctextsize**

Default: 524288

Maximum number of bytes of memory used for the command pane scroll-back buffer. Increase this value if you keep scrolling back only to find that the thing you are interested in has scrolled out of the top of the buffer.

Seconds to wait for debug server before timing out

Format: **servertimeout**

Default: 15

Number of seconds to wait for the debug server before assuming it is dead and disconnecting from it. A number that is too low will sometimes mistakenly disconnect from the debug server, and is not recommended. A fairly high number can be useful for very slow debug servers or debug servers that are being debugged. A high number can be frustrating if the debug server actually does die from time to time, because it keeps the debugger from accepting input while it's waiting to time out.

Editor tab

(Config > Options... > Editor tab)

Reuse editor windows

Format:**openfilesinnewbuffers**

Default: off (files are opened in new editor windows)

Specifies whether a buffer in an existing editor window should be created for files opened from the builder and debugger. If off, a new editor window will be created for each file edited.

Create backup files when saving

Format: **editorbackups**

Default: Off

Automatically create a backup of the on-disk version of a file before saving over it. The backup file has the same name as the original file, with a "~" appended to it.

Drag and drop text editing

Format: **draganddrop**

Default: On

Check to enable drag and drop text editing. After you select a block of text, you can click on it and drag the mouse to move the text to another location.

Tab size

Format: **tabsize**

Default: 8

Number of spaces in a tab when displayed in the Editor.

Indent size

Format: **editindent**

Default: 4

Number of spaces in an indent for languages other than Ada. Used with indentation editor commands. Ada has its own indent setting.

Ctrl+cursor jump size

Format: **editsomesize**

Default: 5

Multiplier used by UpSome, DownSome, LeftSome, and RightSome editor commands. These are bound to the Ctrl+cursor keys by default. So pressing Ctrl+Left will move the cursor left by 5 characters by default.

Configure Editor Buttons...

Opens a dialog which allows editing of the editor buttons using the same format that the editbutton command takes. In the dialog that comes up, there is a list of buttons on the left and a list of available icons on the right.

More Editor Options...

Opens the More Editor Options... dialog which contains additional configuration options for the editor.

Auto Indent Options

Implicit auto indent

Format: **aiimplicitindent**

Default: On

Check to turn on the Editor command AutoIndentImplicit. This will cause the editor to auto indent your file as you type, as opposed to manually invoking the auto indent. Disabled, AutoIndentImplicit has no effect.

Implicit auto indent in comments

Format: **aiimplicitindentincomments**

Default: On

If aiimplicitindent is on, then check to enable AutoIndentImplicit within comments. Disabled, AutoIndentImplicit has no effect in comments.

Switch bodies indented two instead of one

Format: **aiswitchintwo**

Default: On

C - Switch bodies are indented two levels so that case labels are indented one level in from the switch. Disabled, the case labels are even with the switch.

Ada - Select bodies are indented two levels so that when labels are indented one level in from the select. Disabled, the when labels are even with the select.

Indent comments when indenting multiple lines

Format: **aitouchcomments**

Default: Off

Auto indent comments even when auto indenting multiple lines. If Disabled, then comments aren't modified by AutoIndent unless AutoIndent is run on just one line.

Comments stick flush left

Format: **aicommentsstayflushleft**

Default: On

Check to keep comments from indenting away from the left margin. If you want to indent a comment that is stuck to the left margin while this is on, insert a space just before the comment to get it unstuck, then use AutoIndent.

C chars aligned like '*' in comments

Format: **aicharslikestarincomment**

Default: "-"

To have characters other than '*' line up in comments as if they were '*', make them part of this string. For example, this allows correct auto indention of comments which have a column of -'s down the left side, lined up under the * in the /*.

C paren indent mode, Ada paren mode

Format: **aiparenindentmode**

Default: IndentInTwo

Format: **aiadaparenindentmode**

Default: EvenWithParen

Controls how the editor indents a line if it starts within an open paren/close paren pair in the corresponding language.

EvenWithParen - If there is a non-whitespace character between the open paren and the end of its line, then the lines enclosed in () start at the same column as that character. Otherwise, the lines enclosed in () start in the column just after the open paren.

IndentInTwo - The lines enclosed in () start two indent levels in from the open paren's line.

More Editor Options...

(Config > Options... > Editor > More Editor Options...)

Print 2 columns in landscape

Format: **editprint2column**

Default: On

Print files in landscape mode, with two pages per sheet.

Temp file directory

Format: **tempfiledir**

Default: (blank, looks for the TMPDIR, TEMP environment vars, resorts to /tmp)

Directory used for temporary editor files.

Initial width in characters

Format: **editwidth**

Default: 80

Initial width (in characters) of the internal MULTI editor. This option is only useful when the **rememberwindowpositions** option is off. See also “Save window positions and sizes” on page 242.

Initial height in characters

Format: **editheight**

Default: 32

Initial height (in characters, i.e. lines) of the internal MULTI editor. This option is only useful when the rememberwindowpositions option is off. See also “Save window positions and sizes” on page 242.

Selection margin width in pixels

Format: **selectionmarginwidth**

Default: 13

Width of the left margin of the editor. If the width is 0, the left margin does not appear, and text can no longer be selected by line using the margin. Changing the selection margin width does not affect already open editors, only new editors.

Generate auto-recover file every ... seconds

Format: **editincrfrequency**

Default: 120

The editor creates auto-recover files every so often, in case the power goes out or the editor crashes. If this happens, the next time you open the editor on the file it will give you the option to restore to the auto-recover file. This option sets the frequency at which the editor generates these auto-recover files.

Per File Settings Defaults

The settings in this section apply to files when the files are first opened. You can use View > Per File Settings... in the Editor to modify these settings on a per-file basis. However, changes to these settings will also affect already open files, so you have to make the changes on a per-file basis after you make any desired changes to the defaults.

Spaces per indent for Ada

Format: **adaindentsize**

Default: 3

Number of spaces in an indent for Ada files.

Ada continuation line indent

Format: **adacontinuationsize**

Default: 2

Number of spaces in the indent for a continuation line in Ada files.

Word wrap

Format: **wordwrap**

Default: Off

Automatically split lines on word boundaries as you type.

Wrap column

Format: **wrapcolumn**

Default: 79

When wordwrap is on, this is the last column that will be used before wrapping to the next line unless a single word is so long that it can't fit within this many columns, in which case it will be put on a line by itself.

Wrap indent offset

Format: **wrapindent**

Default: 2

When a word wrap to the next line occurs, the line is automatically indented this many extra spaces from where it would normally appear.

Alternate Editor Options

Allows you to use a third-party editor in MULTI.

Use Alternate editor

Format: **usealternateeditor**

Default: Off

When enabled, MULTI opens an external editor instead of using the internal one.

Use xterm for alternate editor

Format: usextermforalternateeditor

Default: Off

When enabled, MULTI runs the alternate editor within a newly created xterm.

Executable

Format: **editor**

Default: me

Name of the alternate editor executable, with full path if the directory isn't in your path.

Command line arguments

Format: **editorlaunch**

Default: (blank)

Command line arguments for the alternate editor, not including the name of the editor itself. There are three special strings in editorlaunch that get replaced with useful information for the alternate editor. "Third party editors" on page 2 contains the required configuration of these three strings for some common editors.

%file0 - Replaced with the first file MULTI is opening.

%line - Replaced by the line number the first file should be opened to.

%files - Replaced by a space separated list of all the files other than the first one. There is no mechanism for conveying what line numbers to open these files to.

Version Control tab

(Config > Options... > Version Control tab)

Use version control

Format: **dontusevc**

Default: Off (On in GUI)

The meaning of this option varies depending on whether you access it through the Config menu or through the command line or a config file.

If you are accessing this option from the GUI, check it to enable version control awareness.

If you are setting this option from the command line or a config file, turn this option on to *disable* version control awareness.

Automatic checkout

Format: **preventautocheckout**

Default: Off (On in GUI)

The meaning of this command varies depending on whether you access it

- through the Config menu
- through the command line
- through a config file

If you are accessing this option from the GUI, check it to enable automatic checkout of files when you start to edit them.

If you are accessing this option via the command line or a config file, turn this option on to *prevent* automatic checkout of files when you try to begin editing them.

Version control system

Format: **versioncontroltype**

Default: MVC

Controls which version control system MULTI uses.

MVC - MULTI Version Control. Green Hills Software provides this version control system as part of MULTI.

ClearCase

RCS

Command

Format: **vc_command**

Default: (varies per versioncontroltype)

Command prepended to all the other version control configuration strings. For instance, if vc_command is "mvc", and vc_checkout is "co", then to check a file out, MULTI would issue "mvc co filename". It is ok to leave this blank if it’s not required.

The version control configuration strings are provided in case your site requires a slightly different syntax. Attempting to set up MULTI to automatically use an

arbitrary unsupported version control system by modifying the settings for, say, ClearCase, may or may not be possible. If you are so inclined, the best hope for success is probably with ClearCase selected. Any changes to the defaults for a system may break support for that version control system (until you restore that system to the default settings).

Under VC

Format: **vc_undervc**

MULTI uses this command to determine if a file is under version control. Only used with ClearCase. If the output of this command begins with the name of the file being checked, the file is considered to be under version control.

Check out

Format: **vc_co**

Command to check out a file from version control for editing.

Get

Format: **vc_get**

Only used for MVC and RCS. Command to get a read only copy of the latest version of a file.

Un check out

Format: **vc_unco**

Command to undo the effects of a check out. Reverts to the latest version of the file in the repository and restores the file's checked-in status (if applicable), discarding any edits since the check out.

Check in

Format: **vc_ci**

Command to check in a file as the latest version.

Check in, no comments

Format: **vc_cinocomments**

Command to check in a file as the latest version without entering any comments into the version control logs.

Create

Format: **vc_create**

Command to put a file under version control for the first time (i.e. have version control start managing a file).

Show history

Format: **vc_show**

Command to output a list of versions a file has gone through, with optional comments or information about each.

Who

Format: **vc_who**

Only used for ClearCase. Command to check which user has this file checked out. Output of this command is the username who has this file checked out.

Previous version

Format: **vc_prevver**

Only used for ClearCase. Used to get version number for the previous version of a file.

Restore Defaults for This VC System

Resets the version control configuration strings to their defaults for this system. This also happens when you switch to another version control system and back.

Colors tab

(Config > Options... > Colors tab)

Colors tab: Global Colors

Background

background: #ffffff

Background color of "user areas" such as the background behind code or text that you enter.

Foreground

foreground: #000000

Default color of text.

Control Area

controlcolor: #c0c0c0

Background color of "control areas" such as menu bars.

Selection

select: #000080

Background color of text selections (foreground color chosen automatically).

Builder File Coloring check box

Format: **colorbuilder**

Default: On

Check to enable coloring the files in the Builder based on their type (code, build file, etc.).

Builder File Coloring settings		
GUI Name	**Command**	**Hex RGB Default**
NoBuild	nobuild	#b00000
Program	program	#900010
SubProject	subprogram	#c80020
Source	source	#00b010
Header	header	#007010
Documentation	documentation	#a0a0a0
Library Build	singlelibrary	#000080
Object	object	#000050
Library	library	#004580
Script	script	#808000
Other	other	#808080

Color of the corresponding type of file in the Builder.

Colors tab: Debugger Colors

Assembly

Format: **assembly**

Default: #0000ff

Color of interlaced assembly code in the Debugger.

Break Dot

Format: **bdotcolor**

Default: #00cd00

Color of break dots in the Debugger; the dots that show where a breakpoint could go.

Status

Format: **breakcolor**

Default: #ff0000

Color of the program status ("STOPPED", "RUNNING", etc.).

Context Arrow

Format: **pointercolor**

Default: #0000ff

Color of the context arrow, which indicates which context to use for commands in the Debugger.

Colors tab: Syntax Coloring

Format: **colorsyntax**

Default: On

Color code according to its syntax.

Syntax Color settings		
GUI Name	**Command**	**Hex RGB Default**
Comments	comment	#008000
Keywords	keyword	#0000ff
Dead Code	deadcode	#808080
Numbers	number	#e000e0
Strings	string	#800000
Characters	character	#c000c0

Color of corresponding item in source code. Dead code refers to code enclosed in #if 0 ... #endif.

Color C++ comments in C

Format: **cppcommentsinc**

Default: On

Color C++ style comments (//) as comments, even if the source file is a C file.

Other Configuration options

This section describes configuration options not accessible from the Config menu.

NOTE: These configuration options are only available from the Debugger Command Pane using the Debugger command **configure** or from a Config file.

clearbuttons

Format: clearbuttons

Removes all the buttons so they can be created from scratch with debugbutton.

debugbutton

Format: **debugbutton** [*num*] [*name*] [[**c=**]*command*] [[**i=**]*iconname*] [[**h=**]*helpstring*] [[**t=**]*tooltip*]

This command adds a new icon button to the debugger toolbar.

command, *iconname*, *helpstring*, and *tooltip* are all either single words, or quoted strings. Quoted string are of the form:

```
"This is a quoted string."
```

There are several forms of the command:

Form	Meaning
debugbutton	By itself, the command lists all the defined buttons. Note that the quit button and the spacer before it are never listed. Those buttons are special and can not be modified or deleted.
debugbutton 0	Deletes all buttons (except the quit button and its spacer).
debugbutton *num*	Deletes the button numbered *num*.
debugbutton *num name* [...]	Replaces the button numbered *num*
debugbutton *name*	Deletes the button named *name*
debugbutton *name* [...]	If a button named *name* exists, the button is replaced. Otherwise a new button named *name* is added to the end of the debugger toolbar.

command is the command executed when the button is pressed. You may use semicolons in the command to execute multiple commands. For example:

```
debugbutton printxy c="print x;print y"
```

iconname is the name of the icon associated with the button. If not specified, then the first letter of the command name will be used as the icon for the button.

iconname may either be the name of one of MULTI's built-in icons (see below for how to obtain a list of these names), or it may be the filename of a bitmap you have created yourself. If the filename is not an absolute filename, it is assumed to be relative to the directory where MULTI is installed.

If you create your own bitmap file, it must end in a **.bmp** extension and must be in the uncompressed 16-color Windows Bitmap format. Other color depths are not supported, and compressed bitmaps are not supported. An easy way to create such bitmaps is to use the Paint accessory under Microsoft Windows, and make sure you choose "16 Color Bitmap" in the "Save as type" drop-down list box of the "Save As" dialog.

The built-in icons in MULTI are 20 pixels wide by 20 pixels tall, so your buttons will look best if you also use this size for your custom bitmaps.

By default, the color light gray in your custom icons will become transparent. You can specify additional color translations for your custom icon by appending a string of the form "oldcolor1=newcolor1&oldcolor2=newcolor2" with a question mark to the end of your bitmap filename. For example:

```
debugbutton Hello c="echo hello"
i="/home/user/hello.bmp?black=fg&dkgray=shadow&white=highlight" h="Say
hello"
```

You can use the following values for oldcolor and newcolor:

Oldcolor (R,G,B values)	Possible values for newcolor
white (255,255,255)	white (default) highlight
ltgray (192,192,192)	ltgray transparent (default)
dkgray (128,128,128)	dkgray (default) shadow
black (0,0,0)	black (default) fg

helpstring is the help text that appears at the bottom of the window when the mouse moves over the button.

tooltip is the tooltip text that appears when you move your mouse over the button and wait. If you do not specify a tooltip, the name of the button will be used.

button

This command is deprecated. Use **debugbutton**. See **debugbutton on page 266**. 1.8.9 MULTI users upgrading to MULTI 2000 should note that the syntax for the **debugbutton** command is different than it was for the **button** command.

cleareditbuttons

Removes all the editor buttons so they can be created from scratch with editbutton.

editbutton

Format: **editbutton** ButtonName c="commands" [i=*iconname*] [h="Status bar string[@@tooltip string]"]

Creates a new editor button. For additional information about creating new buttons, see **debugbutton on page 266**.

ButtonName - Name of the button. If iconifiedbuttons is off, this is the name that goes on the button. If no tooltip string is provided, this becomes the tooltip.

c - Commands the button executes when pressed.

i - Name of icon to use on the button if iconifiedbuttons is on. (See Config > Options... > Editor > Configure Editor Buttons... for a list of icons with pictures.) Alternatively, you can specify your own 16-color bitmap (*.bmp) as the icon. If no icon is specified, the help icon (question mark) is used instead.

h - Status bar string which shows up as soon as the mouse is over the button, and tooltip string which shows up after the mouse hovers over the button for a short time. The status bar string and tooltip string are separated by '@@'.

clearkeys

Removes all keybindings so they can be created from scratch with keybind.

keybind

Format: **keybind**

Format: **keybind** *location*

Format: **keybind** *key*[|*modifiers*][**@***location*][=*command*]

Assigns an action to a key pressed while holding down modifiers while the cursor is in a specified area of a window. In other words, if the mouse is in the specified location (area of a window) and the specified modifiers (such as shift) are held down, and the specified key is pressed, then the specified command will execute. The syntax for this command is essentially the same as the **mouse** command, except **keybind** does not use a click count.

To print the current key press setting for a location, type **keybind** followed by the location (see below for valid locations). If no location is specified, then all settings for the source pane are displayed.

key is a single ASCII (or ISO8859) character or a quoted string containing the name of one of the keys on the keyboard, such as **"BACKSPACE"** or **"F3"**. A list of the acceptable key names are obtainable by typing:

```
keybind "????"
```

Characters needing more than one key press (besides the Shift, Control, and Meta keys) to generate cannot be used. To specify a double quote, put it inside double quotes: `"""` (that is three double quotes in a row).

modifier is any combination of **Shift, Meta, and Control**. If a modifier is specified, then the command is only run if that modifier is depressed at the time the key is pressed. If more than one modifier is specified, they should be separated from each other and from *key* by vertical bars '|'. The BACKSPACE key is different from **h|Control** even though their ASCII representations are the same.

A *location* may be one of the following:

Location	Meaning
All	Any of MULTI's windows, except the Editor.
InputWindow	Any MULTI input window, such as the command pane or a remote or I/O window.
OutputWindow	Any output only window, such as the source pane or a monitor window.
View	Anywhere in a view (data explorer) window, excluding the title bar.
Title	Anywhere in a title bar of a data explorer or monitor window.
Command	Anywhere in the command pane or source pane.
Remote	Anywhere in a 'pass through' or I/O window.
Monitor	Anywhere in a monitor window, excluding the title bar.
Labels	Displays the field names in the data explorer's pane.
Values	Contains the field values in the data explorer's pane.
Name	Region of the title bar displaying the name of the expression in the data explorer, or a monitor window command.
Type	Region of the title bar displaying the type of expression in a data explorer.
Freeze	Region indicating whether the window is frozen.
Close	Close box in the title bar.
Pop	Near the pop arrow on the title bar.
Help	Near the question mark on the title bar.
Menu	Inverted triangle in the data explorer title bars.
Dup	Duplication button, if **togglebuttons** is set.
Shrink	Shrink button, if **togglebuttons** is set.
ScrollBar	Entire scrollbar.
UpArrow	Arrow pointing up at the top of the scrollbar.
DownArrow	Arrow pointing down at the bottom of scrollbar.
ScrollArea	Scrollbar between the arrows.
Thumb	Grey region in scrollbar middle.

Location	**Meaning**
AboveThumb	Blank area above Thumb and below arrow.
BelowThumb	Blank area below Thumb and above arrow.
Edit	MULTI Editor.

If *location* is omitted, the default is **Command**.

These locations are arranged hierarchically with:

Locations hierarchy	
Location	**Composed of:**
All	**InputWindow**, **OutputWindow**, **View**, and **Title**
InputWindow	**Command** and **Remote**
OutputWindow	**Source** and **Monitor**
View	**Labels** and **Values**
Title	**Name**, **Type**, **Freeze**, **Close**, **Pop**, **Help**, and sometimes **Dup** and **Shrink**.
ScrollBar	**UpArrow**, **DownArrow**, and **ScrollArea**
ScrollArea	**Thumb**, **AboveThumb**, and **BelowThumb**

command is any MULTI command. The text it specifies may include any of the following special sequences. These sequences are not valid for key binding in the Editor:

Sequences for *command*	
Sequence	**Meaning**
%s	Replaced by the current selection.
%p	Replaced by the current selection if it exists, otherwise MULTI prompts for input.
%P	Always prompts for input.
%w	Replaced by a special number identifying the window to MULTI.
%x	Replaced by the x location in the window when pressing the button.
%y	Replaced by the y location in the window.
%k	Replaced by the ASCII expansion of the key. For the key labeled **a**, this is "**a**."
%m	Replaced by **Press**.
%%	Replaced by **%**.

If the *key*[|*modifier*]**@***location* of a **keybind** input matches one previously defined, then the new definition replaces the old. If no command is specified in the new definition, then the new definition deletes the old.

When a key is depressed in a window, MULTI searches for keybind actions that match. There may be more than one, in which case MULTI chooses the one whose location is most specific. If there are still several, then MULTI chooses one arbitrarily.

Example

With this command, MULTI evaluates and prints the selection when the first function key is pressed in the source window.

```
keybind “F1”=print %s
```

With this command, the Debugger opens a data explorer which displays the current selection when the first function key is pressed anywhere while the Ctrl key is depressed.

```
keybind “F1”|Control@All= view %s
```

With these commands, MULTI scrolls the window when the up or down arrow is depressed.

```
keybind “Up”@All= (%w) scrollcommand 1l
keybind “Down”@All= (%w) scrollcommand -1l
```

With this command, MULTI single steps the program when Ctrl+s is pressed in the source window.

```
keybind s|Control=S
```

The following is a special case: Any key bound to the command **HELP** at location **All** opens context sensitive help.

```
keybind ?|Control@All=HELP
```

clearmenus

Removes all menus so they can be created from scratch using **menu**.

menu

Format: **menu** *name* **{{***label cmd***}}**
menu *name*
menu

Defines a menu to be attached to a menu bar, MULTI button, mouse button, or key from the keyboard. All menus have a name, by which they are run, and a body which lists labels and associated commands. When a menu is opened, the menu name at the top of the menu is shown with all the labels beneath. Selecting a label causes the associated command to execute.

To choose a menu item, do one of the following:

- Click it.
- Use the arrow keys to highlight it and press Enter.

If one of the characters in a menu label is preceded with an ampersand (&), it will be underlined, and if the user types that character while the menu is up, the command associated with that label will be executed just as if the user had clicked on the label.

MULTI comes with a set of predefined menus for the main MULTI window, as well as for the Editor. (See “Debugger menus” in Debugging with MULTI 2000 and “Editor menus” on page 164)

If you enter **menu** by itself, a list of all defined menu names is printed. Entering **menu** followed by a menu name displays the body of that menu. (This is case sensitive.)

To create a menu, enter **menu** followed by the name you give it, followed by the set of labels and commands. The entire body of the menu must be enclosed in curly braces **{}**, as well as curly braces for each line. The first entry of each line is the label that appears in the menu left justified. The second entry is the command corresponding to the label. (Each of these lines must be contained in its own set of curly braces.) The command portion can contain its own subset of curly braces, such as using the **if...else** command, as long as they are paired correctly. If the command is a single command (not a list of commands), and that command has a key binding associated with it, then that key binding will be displayed (right justified) next to the label.

The following example creates a menu named **RunCmds**:

```
menu RunCmds {
                {Step s}
                {Next S}
                {Run r}
                {Go c}
                {Return cU}
}
```

Menus can contain other menus, for example:

```
menu Main {
                {RunCmds -> RunCmds}
                {Up {E 1}}
                {Down {E -1}}
                {ToPC E}
```

Invoking this menu and moving the cursor to the right edge of the **RunCmds** entry calls up the submenu **RunCmds**.

The following example allows you to customize your own menu and bind it to a mouse click . Typing the following in the Debugger command pane allows you to use your customized menu in the Debugger with a click.

```
menu MyMenu {
{Go C}
{Step s}
{Next S}
}

mouse mouse*Press1@All=->MyMenu
```

To replace a menu definition, type a new menu command with the same name.

To edit existing menus, choose Config > Options... > General tab > Menus.... You can also use this dialog box to add or delete menus. See also “Menus...” on page 245.

Opening menus

You can open a menu by typing **->** directly before their name in the Debugger command pane. For example, to open the menu **Main** from the command window, enter:

```
->Main
```

To bind a menu to a Debugger button, type:

```
debugbutton RunCmds ->RunCmds
```

To bind a menu to a mouse button, enter:

```
mouse mouse3*Press1@All=->RunCmds
```
(where `mouse3` is the rightmost mouse button)

clearmice

Removes all mouse button bindings so they can be created from scratch with **mouse**.

mouse

Format: **mouse** *location*

Format: **mouse mouse** *button_num*[AtOnce][***click** *click_num*] [|*modifiers*] [**@***location*] [=*command*]

Defines the function of the mouse buttons. The first form of the command prints the current mouse commands in the given location (see the list below for valid locations), and the second form of the command changes the way the mouse buttons work.

To execute the desired command, you must indicate a combination of modifiers, button number, number of clicks, and a location for the command to occur, as explained below.

mouse *button_num* may either be the keyword **Any**, meaning any mouse button, or the word **mouse** followed by some of the digits between 1 and 5, meaning those mouse buttons whose numbers are listed. Not all mice have five buttons, so any commands assigned to non-existent buttons are not run. For example, to set a command for buttons **1** or **3**, use **mouse13**.

The click count option, ***click** *click_num*, may be omitted, in which case it defaults to one click. However if present, it specifies the number of times the mouse button must be depressed before the command is executed. *click_num* may be a number between 1 and 5. The keyword **click** may be replaced by **press** in which case the command executes on the button press, rather than on the release. Or it may be **either**, in which case the command executes on both the press and release.

The keyword **click** is also followed by the text **(AtOnce)**. This means the command bound to it executes immediately rather than pausing briefly to see if this is part of a click sequence. It has the unfortunate side effect that this command is always done immediately, even if followed by the second click of a double click. This is acceptable for many options such as the standard selection clicks: one click sets the insertion point, two clicks select the current word, three the line, and so forth. The downside to not using AtOnce is that there is a delay when invoking single-click commands (while it's waiting to see if there will be the second click of a double click, etc.).

modifiers may be one of the keywords **Shift, Meta,** or **Control**. All modifiers are preceded by a vertical bar "|". This separates them from each other and from **mouse***button_num*. If a modifier is specified, then the command is only run if that modifier is depressed. If more than one modifier is specified, then all modifiers listed must be depressed simultaneously.

A *location* is one of the following:

Location	**Meaning**
All	Any of MULTI's windows.
InputWindow	Any MULTI input window, such as the command pane or a remote pass through window.
OutputWindow	Any output only window, such as the source pane or a monitor window.
View	Anywhere in a view (data explorer) window, excluding the title bar.
Title	Anywhere in a title bar of a data explorer or monitor window.
Command	Anywhere in the command pane.
Remote	Anywhere in a pass through or I/O window.
Source	Anywhere in the source pane.
Monitor	Anywhere in a monitor window, excluding the title bar.
Labels	In the pane of a data explorer displaying the field names.

Location	**Meaning**
Values	In the pane of a data explorer containing the field names.
Name	Region of the title bar displaying the name of the expression in the data explorer, or the command of a monitor window.
Type	Region of the title bar displaying the type of expression in a data explorer.
Freeze	Region indicating whether the window is frozen.
Close	Close box in the title bar.
Pop	Near the pop arrow on the title bar.
Help	Near the question mark on the title bar.
Dup	Duplication buton, if **togglebuttons** are set.
Shrink	Shrink button, if **togglebuttons** are set.
ScrollBar	Entire scrollbar.
Menu	On the inverted triangle in data explorer title bars.
Edit	MULTI Editor.

If *location* is omitted, then Source is the default.

These locations are arranged hierarchically with:

Locations hierarchy	
Location	**Composed of:**
All	**InputWindow**, **OutputWindow**, **View**, and **Title**
InputWindow	**Command** and **Remote**
OutputWindow	**Source** and **Monitor**
View	**Labels** and **Values**
Title	**Name**, **Type**, **Freeze**, **Close**, **Pop**, **Help**, and sometimes **Dup** and **Shrink**.

command is any MULTI command. The text specified may include any of the following special sequences. These sequences are not valid when binding commands in the Editor:

Sequences for *commands*	
Sequence	**Meaning**
%s	Replaced by the current selection.
%p	Replaced by the current selection if it exists, otherwise MULTI prompts for input.
%P	Always prompts for input.
%w	Replaced by a special number identifying the window to MULTI.
%x	Replaced by the x location in the window when pressing the button.
%y	Replaced by the y location in the window.
%k	Replaced by the null string.
%m	Replaced by **Press** or **Release,** as appropriate.
%%	Replaced by **%**.

If the **mouse***button_num*[|*modifiers*>][**@***location*] of a mouse input matches one previously defined, then the new definition replaces the old one. If no command is specified, then the new definition deletes the old.

When the mouse is clicked in a window, MULTI searches its list of mouse actions to match. If there is more than one, the following procedure determines which one to choose: First, the one whose location is the most specific. If there are still several, then the one which accepts the fewest number of buttons is chosen. If there are still several, then MULTI chooses one arbitrarily.

Example

```
mouse mouse1=print %s
```

The Debugger evaluates and prints the selection when you click the left mouse button once in the source window.

```
mouse mouse1*Click2@All=view %s
```

The Debugger opens a data explorer displaying the current selection when you double-click the left mouse button anywhere.

configurefile
configure

Formats: **configurefile** filename
configure filename

Used in a config file (.cfg file), these both read filename in as a .cfg file. Then processing of the original .cfg file continues as normal. **configure** used in this capacity is deprecated. Use **configurefile** instead.

grabtimeout

Format: **grabtimeout**

Default: **-1**

If *time* is less than zero, then MULTI does not check to see if there are any outstanding grabs on the X server each time it stops. Otherwise it checks, and if both the keyboard and the mouse are grabbed, waits *time* is seconds before aborting the grab and debugging. This is useful for debugging X-windows programs that are broken and keep grabbing the keyboard and mouse, making it impossible to issue commands to the debugger.

clickpause

Format: clickpause

Default: **4**

Specifies the length of time, in tenths of a second, that MULTI waits between button presses to get a double or triple click. For example, if *time* is four, and two clicks come within four tenths of a second of one another, and they are on the same button in the same place, then they are treated as a single double-click. On the other hand, if they come with more than four tenths of a second between them, then they are treated as two single-clicks. If MULTI determines there is no possible double click command, then it does not wait.

viewdef (Data Explorer Window Format)

Format: **viewdef**

Default: *formats*

Default setting: **ShowName, ExpandValue, ReEvalContext, ShowChanges**

This resource sets which items in the format menu of the data explorer windows are set by default. *formats* is a list containing the following keywords:

Default data explorer formats	
ShowAddress	ShowAllFields
ShowName	ShowBases
Alternate	ReEvaluate
OnlyAlternate	ReEvalInGlobal
Hex	ReEvalInContext
Oct	UseAddress
ExpandValue	ShowChanges

For more information on the format menu, see “Data explorer format menu” in Debugging with MULTI 2000.

geometry

Format: **geometry**

Default: *width***x***height*+*x_offset*+*y_offset*

Default setting: Depends on screen size and varies from system to system. The width is approximately wide enough to display 80 characters on a line.

Sets the size and position of the Debugger window. The *offset* values are optional. The italicized fields are specified in pixels. The “+” field may also be “-” to specify relative to lower and right edges. For example:

```
geometry 500x700+0+0
```

usewmpositioning

Format: **usewmpositioning [on | off]**

Default: Off

Allows the window manager to decide where all windows should go. When off, MULTI tries to be a bit smarter about where windows should go.

iconify

Format: **iconify**

Default: **Off**

Specifies whether the next MULTI window will come up iconified. *state* may be **on** (iconify) or **off** (do not iconify). This option is reset to off when the icon for the iconified window appears.

ignoremotion

Format: **ignoremotion**

Default: **4**

Specifies the number of pixels of movement in the mouse that MULTI ignores during the time of pressing and releasing a mouse button. If the mouse is moved by more than the value of *pixels* between press and release, then the mouse click is no longer treated as a single click.

linesnonoverlapped

Format: **linesnonoverlapped**

Default: **4**

By default, when the Debugger opens a data explorer or monitor window using its own positioning algorithm, it tries to stack it on top of previous windows. To save screen space, it overlaps the new data explorer or monitor window on top of the old one. This obscures the bottom of the previous window. Set **linesnonoverlapped** to be the number of lines of the old window that should still be visible after the new window is placed on top of it.

editparenmatch

Format: **editparenmatch**

Default: 10

Every time you type a right parenthesis, right square bracket, or right curly brace, the Editor briefly selects the matching one. This controls how long it pauses on the selection. *time* is given in tenths of a second.

sharedsymbols

Format: **sharedsymbols**

Default: **On**

Determines whether **dblink** is run to process debug information from shared libraries. This is also controlled with the **-noshared** Green Hills compiler option.

procqualifiedlocalimpliesoutermostblock

Default: Off

On - Even if the context pointer is in an inner block, and that inner block defines a variable which has the same name as a variable in the outer block, a reference to that name in that procedure will reference the outer variable of that name.

Off - If the context pointer is in the inner block and the variable name is referenced, the inner variable will be used.

warnonbpreplacement

Default: Off

To get a warning from the debugger before replacing a breakpoint that was already there, turn this option on. This is useful to avoid losing a long breakpoint command by accidentally replacing that breakpoint with a new breakpoint (which has no command).

warnoncmdadrlinepromotion

Default: Off

To get a warning when setting a breakpoint on a line with no corresponding assembly (no breakdot), turn this option on.

attempttoshowoldversionofupdatesource

Default: Off

To have the debugger attempt to show the versions of source files that were used to build the executable being debugged (via version control), turn this option on.

allowexecutioninbpcommand

Default: Off

To allow stepping, nexting, and execution of command line procedure calls from within a breakpoint command, turn this option on. This is somewhat risky, in that if the execution from within the breakpoint command causes another breakpoint to be hit (or the same one), infinite breakpoint command recursion can occur. Continue ("c") is always allowed in a breakpoint command.

keeploaders

Default: Off

To have the debugger keep certain item chooser windows (such as file choosers) open after they've been used to load one file, turn this option on. With this option off, such windows disappear after being used once.

icongeometry

Default: 32x64+0+0

icongeometry *width***x***height+x_offset+y_offset*

Specifies the geometry of the iconified form of the debugger. Some window managers ignore this setting.

exprcasesensitivity

Default: Language_Default

Controls the case sensitivity of expression evaluation.

Language_Default - Recommended. Follows the case sensitivity rule for the language of the program being debugged.

On - Expression evaluation is case sensitive.

Off - Expression evaluation is not case sensitive.

gotohitsbpattargetaddress

Default: Off

When using the debugger command **g**, if this option is on, any breakpoint at the destination will be hit as soon as execution begins at the new location. If the option is off, the breakpoint will not be hit.

disasmstyle

Default: remote

Controls the style of Motorola 68000 series assembly code.

remote - XORmacs style if and only if the code is destined for execution on an embedded processor (not going to be executing on the same processor as MULTI).

XORmacs - Always XORmacs style (MOVE.L (12,A6),D0).

unix - Always unix/sun style (movl a6@(12),d0).

synchronous

Default: Off

To enable synchronous X-windows mode, turn this option on. Synchronous mode insures that X-windows calls within MULTI complete before they return. This can be useful when running MULTI on faulty X-servers.

builderposition

This command is deprecated.

Format: **builderposition**

Default: 0x0

Initial position of the builder window from the upper left of the screen in characters and lines. To specify in pixels, append the letter 'p'. This option is only useful if the **rememberwindowpositions** option is off. See also "Save window positions and sizes" on page 242.

minwindowsize

This command is deprecated.

Default: 51x6

Minimum initial size of a non-view window, such as an IO, Target, or Monitor window. If left unspecified, the Debugger will auto-size them appropriately. This option is only useful if the **rememberwindowpositions** option is off. See also "Save window positions and sizes" on page 242.

maxwindowsize

This command is deprecated.

Default: 128x20

Maximum initial size of a non-view window. This option is only useful if the **rememberwindowpositions** option is off. See also "Save window positions and sizes" on page 242.

nodecoration

Format: **nodecoration**

Default: **Off**

If this is **on** and the window manager supports it, then all windows appear without title bars.

QuietTogCmd

Default: **off**

Causes the tog command to not echo the status of the breakpoint(s) it toggles.

Example:

```
D; // delete all breakpoints
b; // set a breakpoint
tog off; // disable the current breakpoint
B; // list all breakpoints
configure QuietTogCmd on
tog on; // enable the current breakpoint
B; // list all breakpoints
```

Appendix

A

Third party tools

This appendix contains:

- Third party version control systems
- Third party editors
- Using the Editor with third party tools
- Using the Debugger with third party tools

Third party version control systems

You can configure MULTI to use a third party version control system, such as RCS or ClearCase. See Chapter 4, “Version control” for information. If you are using a third party version control system other than RCS or ClearCase, MULTI does not have built-in support for it, but you can provide partial support yourself by creating menus which open your version control system as a shell command. For more information on invoking shell commands from editor menus, see “Using the Editor with third party tools” on page 3.

Third party editors

You can configure MULTI to use another editor in place of MULTI’s built-in Editor with the configuration strings: **usealternateeditor**,**usextermforalternateeditor**, **editor** and **editorLaunch**. These options are accessible from the GUI by choosing Config > Options... > Editor Tab > More Editor Options.... For more information on these variables, see Chapter 9, “Configuring and customizing MULTI” or Chapter 10, “Configuration commands”. **usealternateeditor** (which corresponds to the "Alternate editor" checkbox in the GUI) means that MULTI should use a third party editor instead of the build-in editor. **usextermforalternateeditor** (corresponds to "Use Xterm for alternate editor" checkbox) means that MULTI will launch the editor inside an xterm window; use this option if your editor is non-graphical. **editor** (corresponds to the "Executable" text field) should be set to the command name of the third party editor, and **editorLaunch** (corresponds to "Command line arguments" text field) to a string which indicates the format of arguments that the Editor expects. Within **editorLaunch**, the following special escape sequences are recognized:

%LINE The line number.

%FILE0 Filename (first file name if there are multiple ones).

%FILES Where the rest of the file names should go if the editor can accept multiple file names.

For example, here are configurations for commonly used editors:

vi

```
usealternateeditor:    On
usextermforalternateeditor:    On
editor:         vi
editorLaunch:  "+%LINE %FILE0"
```

emacs

```
usealternateeditor:      On
usextermforalternateeditor:    Off
editor:           emacs
editorLaunch:    "+%LINE %FILE0 %FILES"
```

notepad

```
usealternateeditor:      On
usextermforalternateeditor:        Off
editor:         notepad
editorLaunch:   "%FILE0"
```

Depending upon your PATH environment variable, you may need to specify the full path to the Editor.

Using the Editor with third party tools

The Editor can be configured to launch third party tools from buttons or menus. The Editor commands **!**, **Shell** and **CommandToWindow** are appropriate for this purpose.

! (and the equivalent command **ExecuteCmd**) takes the current selection in the editor, if any, and pipes it to the standard input of the specified command, and then replaces the selection with the standard output of the specified command. If there is no selection, the command gets no input, and the output is inserted at the cursor.)

Shell runs a shell command with no input/output redirection.

CommandToWindow runs a shell command and opens a new editor window which the standard output of the command is redirected to.

For more information on **!**, **Shell**, and **CommandToWindow**, see Chapter 7, “Editor commands”.

Here is an example configuration that runs some simple UNIX commands:

```
menu: MyTools {{’insert pretty date’ ! date "+%l:%M %p on
%B %e, %Y"} {’count words’ CommandToWindow "printf
%8s%8s%8s%9s\\\\n lines words chars filename ; wc %FILE"}}
menu: EditMenuBar {{&File ->EditFile} {&Edit ->EditEdit}
{V&iew ->EditView}{&Block ->EditBlock} {&Tools
->EditTools} {MyTools ->MyTools} {&Version ->EditVersion}
{&Config ->Config} {&Help ->EditHelp}}
```

Recall that an easier way to configure menus is to use the Menus... button on the General tab of the Config > Options... dialog.

Using the Debugger with third party tools

External tools can be run from within the Debugger using the **Shell** command. Menus and buttons can be configured to run arbitrary external commands and dynamically construct command-line arguments to those commands. The main facility for constructing command-line arguments is the **%EVAL** escape sequence.

For example, here is a set of configuration directives appropriate for invoking **SNiFF+** from within the Debugger. The variable `_SELECTION` is a Debugger special variable, corresponding to the current selection in the Debugger source pane.

```
shellconfirm:  Off
usealternateeditor:   On
usextermforalternateeditor:  Off
editor:  sniffedit
editorLaunch:  %FILE0
debugbutton:  Class i=letter_c h="SNiFF browse class"
c="shell sniffaccess
browse_class \\*/ %EVAL{_SELECTION}"
debugbutton:  Hierarchy i=dsndnt h="SNiFF hierarchy"
c="shell sniffaccess
hierarchy \\*/ %EVAL{_SELECTION}"
debugbutton:  Symbol i=search h="SNiFF find symbol"
c="shell sniffaccess
find_symbol \\*/ %EVAL{_SELECTION}"
debugbutton:  Retriever i=letter_r h="SNiFF retriever "
c="shell sniffaccess
retrieve \\*/ %EVAL{_SELECTION}"
menu:  Sniff {{Class shell sniffaccess browse_class \*/
%EVAL{_SELECTION}}
{Hierarchy shell sniffaccess hierarchy \*/
%EVAL{_SELECTION}}  {Symbol shell
sniffaccess find_symbol \*/ %EVAL{_SELECTION}}  {Retriever
shell sniffaccess
retrieve \*/ %EVAL{_SELECTION}}}
menu:  MultiMenuBar {{ &File ->FileMenu } { &Debug
       ->DebugMenu } { &View ->ViewMenu } { &Browse
       ->BrowseMenu } { T&arget ->Target } { &Tools
       ->ToolsMenu } { &SNiFF ->Sniff } { &Config
       ->ConfigMenu } { &Help ->HelpMenu } }
```

Index

Symbols

Numerics

A

Index

B

C

Index

D

E

F

Index

G

H

I

J

K

L

M

Index

N

O

P

Q

R

S

Index

T

Index

U

V

W

X

Y

Z