Getting Started with MULTI® 2000 for MCore

UNIX®

DISCLAIMER

PubID: Q56U-I1299-20NG
December 20, 1999 1:15 pm

Welcome to MULTI!

Before you get started:

Notice the graphic insert on pages G-1 through G-4. (Right in the middle of the book.) Grab this insert and **pull it out**. This insert is a graphic version of this tutorial. Use it to help you through the steps on the following pages.

Chapter

1

Up and Running in 10 Quick Steps

1.Create and cd to a new directory

2.Start MULTI

3.Add a [program] object to default.bld

4.Navigate into hello.bld

5.Add a C source file and link file to hello.bld

6.Create the hello.c source file

7.Compile hello.c

8.Connect to a Board

9.Start the Debugger

10.Run some code

The following 10 steps guide you through building and debugging a **Hello World** program using MULTI. These steps give you a feel for how to use MULTI and put into context the information in the *Building and Editing with MULTI 2000 manual* and the *Debugging with MULTI 2000* manual. Attached is a graphical representation of these 10 steps. You may refer to this chart as you go through each step. “Get the details” on page 5 provides a more in depth version of these steps. “Notes for the Pro” on page 15 addresses additional issues and suggestions.

1. Create and cd to a new directory

Creating a new directory simply provides a convenient place to create and store your program. Create a temporary directory called **hello** and **cd** to it.

```
% mkdir hello
% cd hello
```

2. Start MULTI

You can start MULTI by entering **multi default.bld** if the Green Hills tool directory is in your PATH environment variable. If not, then specify the full path name (e.g.: /usr/green/multi). MULTI starts in the current working directory and, without any command-line parameters, will attempt to open the last build file that was open. Adding **default.bld** to MULTI will override this behavior. If **default.bld** is not found, MULTI will create it.

```
% multi default.bld
```

3. Add a [program] object to default.bld

To add a `[program]` object, click the Add button () and enter **hello.bld**.

4. Navigate into hello.bld

To navigate into **hello.bld**, double-click **hello.bld** in the Project pane.

5. Add a C source file and link file to hello.bld

To add a C source file to **hello.bld**, click the Add button () and enter **hello.c**.

6. Create the hello.c source file

1) Double-click **hello.c** in the Project pane.

2) Enter the following text into the MULTI Editor:

```
#include <stdio.h>
int main (int argc, char **argv) {
   printf ("Hello World!\n");
   return 0;
}
```

3) Click the Save and Close button () to save the file and exit the Editor.

7. Compile hello.c

Click the Build button () in the Builder.

The Progress Window shows compilation results for each module. You can close this window after successful compilation.

8. Connect to a Board

Click the Connect button () and enter **pbugserv -mmc2001 com1** into the remote field. Or, enter **simmcore.** Two small windows (the Target and I/O window) will appear when the connection is established. If you are connecting to a hardware board, see “Connect to a Board” on page 10.

9. Start the Debugger

To start the Debugger, click the Debug button () in the Builder. This loads the **hello** program executable into the MULTI Debugger and the program source code is displayed.

10. Run some code

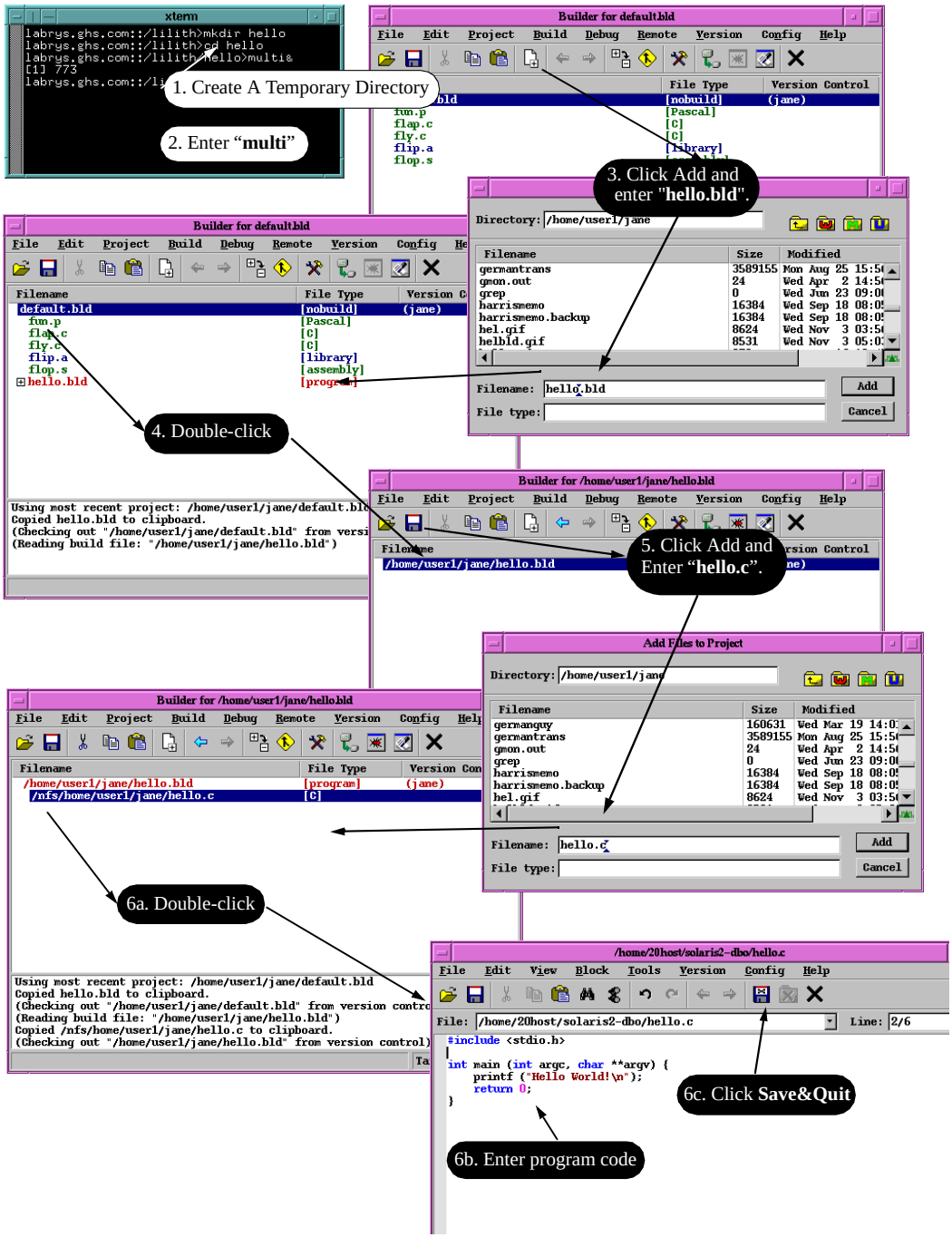
Run some code by clicking the Next button () in the Debugger. This button executes one line of source code at a time. After clicking the Next button () button a few times, you’ll see **Hello World!** appear in the I/O window described in Step 8.

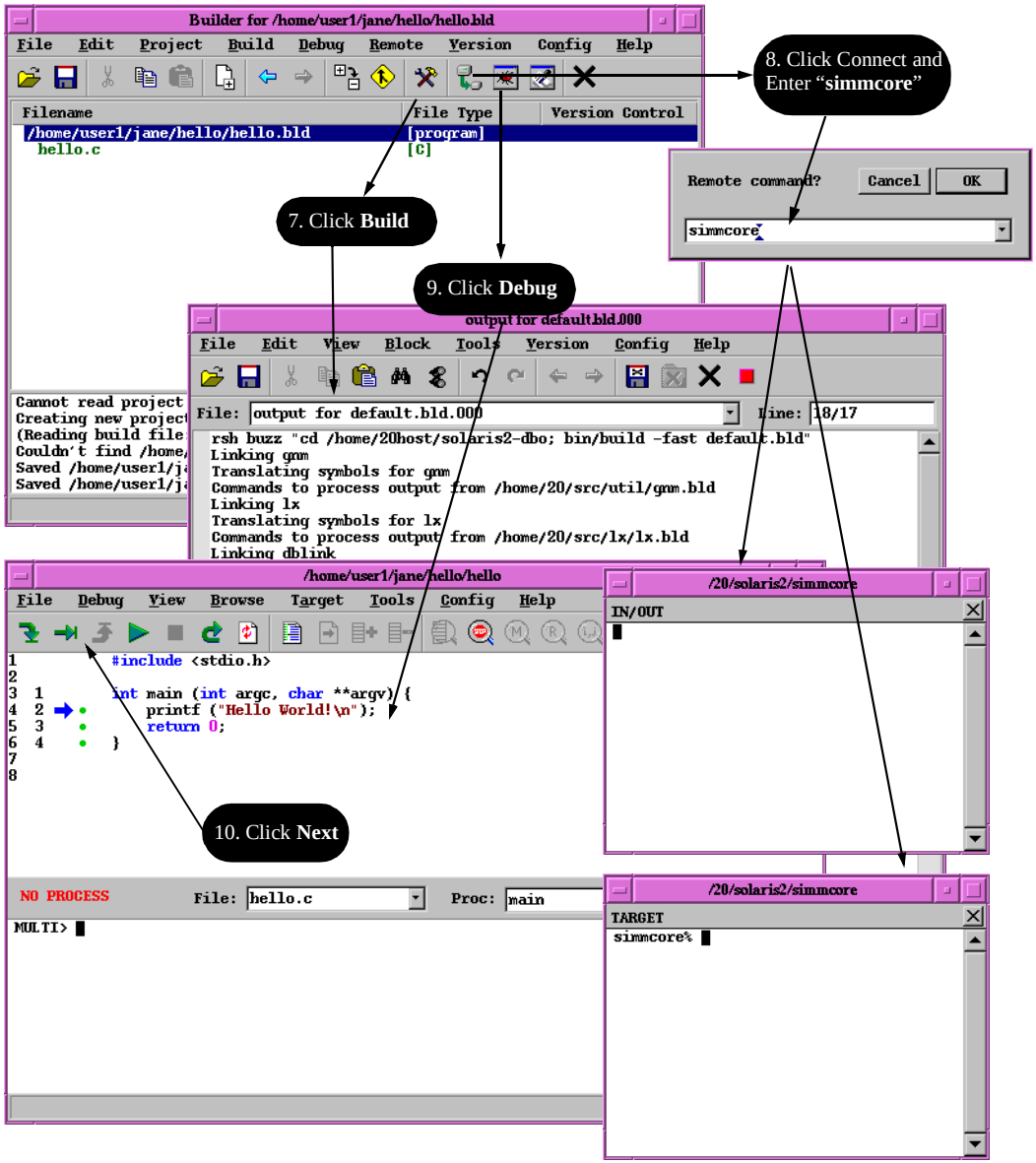
With these 10 simple steps, you have created a MULTI project, written, compiled, and run a program, and produced program output on the host. Progressing to larger applications with multiple program elements is easy. Each element is built and run the same way. Consult the *Building and Editing with MULTI 2000* manual and the *Debugging with MULTI 2000* manual for more information on MULTI's many capabilities.

Introduction to MULTI® 2000 Graphic Insert

This insert contains a graphic representation of the steps presented in Chapters 1 and 2.

To remove insert, grasp pages G-1 through G-4 firmly and pull out.


xterm
labrys.ghs.com::/lilith>mkdir hello
labrys.ghs.com::/lilith>cd hello
labrys.ghs.com::/lilith/hello>multi&
[1] 773
1. Create A Temporary Directory
2. Enter "multi"
Builder for default.bld
File Edit Project Build Debug Remote Version Config Help
3. Click Add and enter "hello.bld".
Add Files to Project
Directory: /home/user1/jane
Filename: hello.bld
File type:
Add
Cancel
4. Double-click
Builder for /home/user1/jane/hello.bld
5. Click Add and Enter "hello.c".
Filename: hello.c
6a. Double-click
/home/20host/solaris2-dbo/hello.c
#include <stdio.h>
int main (int argc, char **argv) {
printf ("Hello World!\n");
return 0;
}
6b. Enter program code
6c. Click Save&Quit


Builder for /home/user1/jane/hello/hello.bld
File Edit Project Build Debug Remote Version Config Help
Filename
File Type
Version Control
/home/user1/jane/hello/hello.bld
[program]
hello.c
[C]
8. Click Connect and Enter "simmcore"
Remote command?
Cancel
OK
simmcore
7. Click Build
9. Click Debug
output for default.bld.000
File Edit View Block Tools Version Config Help
File: output for default.bld.000
Line: 18/17
rsh buzz "cd /home/20host/solaris2-dbo; bin/build -fast default.bld"
Linking gnm
Translating symbols for gnm
Commands to process output from /home/20/src/util/gnm.bld
Linking lx
Translating symbols for lx
Commands to process output from /home/20/src/lx/lx.bld
Linking dblink
/home/user1/jane/hello/hello
File Debug View Browse Target Tools Config Help
#include <stdio.h>
int main (int argc, char **argv) {
printf ("Hello World!\n");
return 0;
}
/20/solaris2/simmcore
IN/OUT
10. Click Next
NO PROCESS
File: hello.c
Proc: main
MULTI>
/20/solaris2/simmcore
TARGET
simmcore%

Chapter 2

Get the details

This Chapter Contains:

- An in depth review of 10 steps for using MULTI

The following reviews the 10 steps introduced in Chapter 1. These discussions include hints, suggestions, warnings, caveats, pitfalls, and frequently asked questions. Please refer to the *Building and Editing with MULTI 2000* manual for more information.

1. Create and cd to a new directory

Creating a new directory simply provides a convenient place to create and store your program. Create a temporary directory called **hello** and **cd** to it.

```
% mkdir hello
% cd hello
```

When you create a temporary directory, you provide yourself a clean working directory for this exercise.

2. Start MULTI

You can start MULTI by entering **multi** if the Green Hills tool directory is in your PATH environment variable. If not, then specify the full path name (i.e.: /usr/green/multi). MULTI starts in the current working directory and looks for **default.bld**. If **default.bld** is not found, MULTI will create this file.

```
% multi
```

- If you are on a command line, you can use either of the following methods:

1. Enter **multi** if **multi** is in your PATH environment variable.
2. Specify an absolute path like /usr/green/multi. In this case, MULTI will find all of its necessary sub-components in the directory in which MULTI resides.

- Generally, it is *not* a good idea to start MULTI from the green directory. Since MULTI does not write to any files in the green directory (with the exception of a **.cfg** file used for preferences), you can keep the green directory clean and stable.

3. Add a [program] object to default.bld

To add a `[program]` object, click the Add button () and enter **hello.bld**.

- You may refer to *The MULTI Builder* on page *16* for information about the Builder.
- In this step, you're basically creating one program called **hello** in your project, **default.bld**.

4. Navigate into hello.bld

To navigate to **hello.bld**, double-click **hello.bld** in the Project pane.

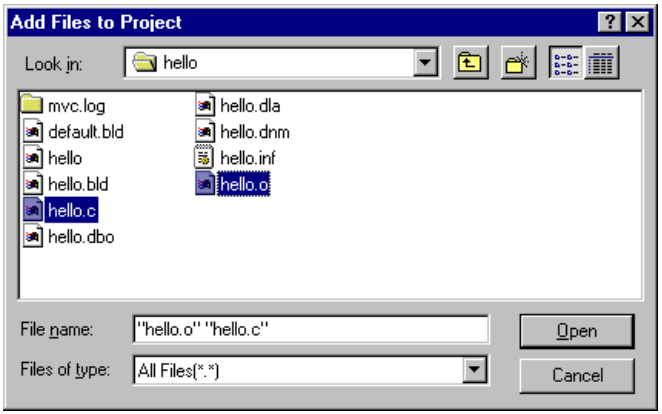
When you navigate to a **.bld** file, you are essentially opening that **.bld** file like an Editor opens a text file. The **.bld** file currently open appears at the top of the Project pane. When a build/compile option is set, that option is stored in the open **.bld** file.

5. Add a C source file to hello.bld

To add a C source file to **hello.bld**, click the Add button () and enter **hello.c**.

If you enter the name of a file that does not exist, MULTI adds the name to the Project pane. The file does not actually exist on disk until you edit it.

- If you enter the name of a file that currently exists, MULTI adds it to the Project pane.
- To add multiple files to a project choose **Edit > Add Files to Project**. The following window will appear. Select multiple files using Shift or Control



6. Create the hello.c source file

a) Double-click **hello.c** in the Project pane.

b) Enter the following text into the MULTI Editor:

```
#include <stdio.h>
int main (int argc, char **argv) {
   printf ("Hello World!\n");
   return 0;
}
```

c) Click the Save and Close button ( ) to save the file and exit the Editor.

Refer to *Building and Editing with MULTI 2000* for further information on launching an alternate editor and advanced forms of control over your alternate editor, such as opening files to a specific line number.

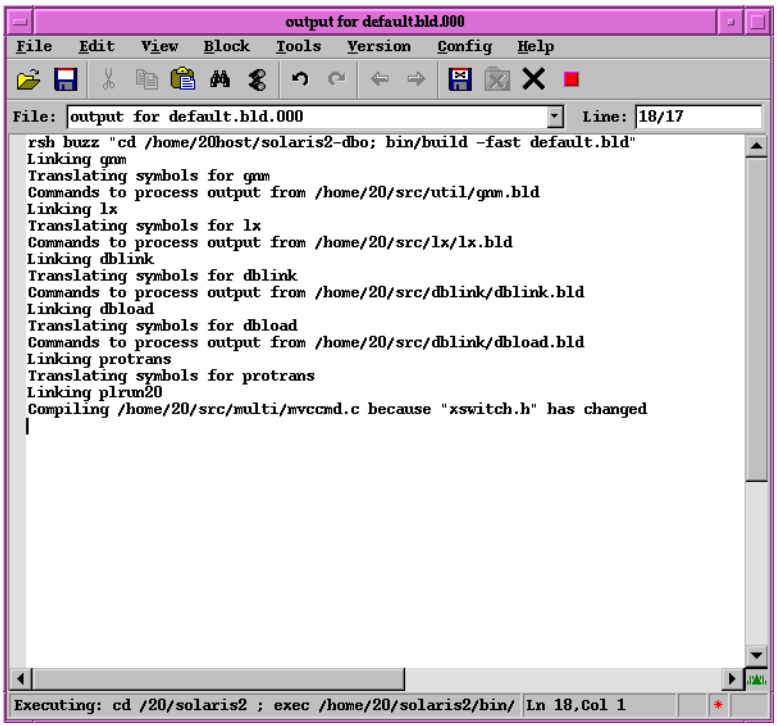
7. Compile by clicking the Build button in the Builder

Click the Build button ( ) in the Builder.

The Progress Window shows compilation results for each module. You can close this window after successful compilation.

- When you click the Build button (), the MULTI Builder will invoke several tools (such as C/C++/EC++/FORTRAN/Pascal/Ada95 compilers, librarian, linker, etc.) to build the project/program you have open in the Builder.
- The MULTI Builder does dependency checking. For example, if you change one item such as the header file, only those affected source files are recompiled, executables are relinked, libraries are re-archived, etc. Changes to the **.bld** files themselves (i.e. setting of build/compile options) are also considered during dependency checking.
- The Build button () is equivalent to **Build > Build** *current_file* in the Builder's menu bar. **Build > Cleanup Intermediate Files** and **Rebuild All** are used just like **make clean** and **make all** are when using makefiles.

- A Builder Progress window displays the build progress.



When the compiler encounters a syntax error you can double-click that error and an Editor window takes you to the line where the error occurred.

- If the Builder Progress window is no longer needed, close it by clicking the Exit button ( ).

8. Connect to a Board

Click the Connect button ( ) and enter **pbugserv -mmc2001 com1** into the remote field. Or, enter **simmcore**. Two small windows (the Target and I/O

window) will appear when the connection is established. If you are using a different target board, see the following.

- MULTI uses a debug server to connect to a target. A debug server is the software module that provides an interface between MULTI and a specific target hardware environment. Green Hills supports a large variety of debug servers that allow connection to many different targets in many different ways. For example, the OCDserv debug server drives your PC's parallel port to connect to a board with on-chip debugging using a Wiggler, while HPserv talks across an ethernet to a HP Processor Probe that connects to the target's JTAG port, and so on. In short, the MULTI environment looks and behaves the same, while the debug server takes care of the uniqueness of your target connection.
- Pre-build demos for use with several boards are provided with your distribution. These demos are complete with setup files and link files. For more information on these demos look at the readme.txt file located in /usr/green/examples.
- In order to use many of the commands below you will need to copy the appropriate setup.ocd or setup.dbs file from /usr/green/examples/*boardname* into your project directory. You should also copy the link file (*boardname*.lnk) and add it to your project to ensure that your program is appropriately linked.
- If you have a hardware board, follow the instructions included with the board's documentation for properly installing the board. Connect to the board by clicking the Connect button  and entering the remote command from the following table. These commands are generic and may not work with all configurations. For specific server options, consult the Green Hills documentation for your board.

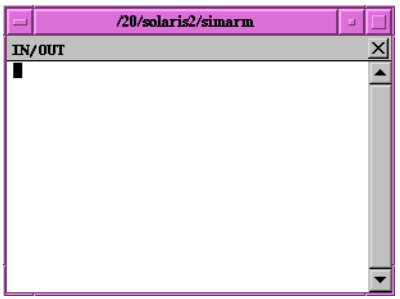
Interface	Supported Board	Command
MCore Simulator		simmcore
PBUG Monitor	MMC2001	pbugserv -mmc2001 com1

- When a debug server establishes connection to a target, two windows appear:

1. The **Target Window** provides an interface to the debug server. In many cases, you do not use this window. Under special circumstances, it provides

a handy interface for lower-level functions. For example, when using the HPserv debug server, the Target window provides a Telnet interface to the Emulator. You can then interface with the emulator directly.

2. The **I/O Window** provides **stdin**/**stdout** to the target. For example, when **printf** is called from the target, the output is displayed in this window. This window works with the standard ANSI IO libraries provided by GHS. If you have your own IO routines, the debug server IO window is ignored.



- For this example, you used an instruction set simulator. The GHS instruction set simulators are intended to provide instruction set valid simulation. Some of them provide limited forms of cycle accurate and cache simulation. However, extensive hardware simulation (including on-chip and on-board peripherals) is not the intention of the simulators. For true (and accurate) co-simulation, refer to third party co-verification tools integrated with MULTI.

9. Start the Debugger

To start the Debugger, click the Debug button  in the Builder. This loads the **hello** program executable into the MULTI Debugger and the program source code is displayed.

10. Run some code

Run some code by clicking the Next button  in the Debugger. This button executes one line of source code at a time. After clicking the Next button  button a few times, you'll see **Hello World!** appear in the I/O window described in Step 8.

- Using all of the MULTI Debugger features is beyond the scope of this tutorial. Please see the *Debugging with MULTI 2000* manual for a complete list of capabilities.
- Some interesting capabilities in MULTI are:

1. MULTI uses the concept of "single-click something to see it, and double-click to bring it up in a separate **Data Explorer** window". It is worth trying this with each of the following:

- variables
- functions
- procedures
- methods
- types
- instances
- classes
- constants
- expressions
- pointers (within a data explorer window)
- nested data structures (within a data explorer window)
- parent classes (within a data explorer window)
- method list (within a data explorer window)
- functions (within the call stack, static and dynamic calls graphs)
- classes (within the class browser)

2. The Assembly button  is useful for displaying source code and assembly interlaced. The Step button  and the Next button  are relative to the assembly code when in interlaced mode.

3. You'll notice green dots , known as breakdots, to the left of each line of executable code. These breakdots show you what is an executable and what is not. You can set breakpoints by simply clicking the breakdot. A red "Stop Sign" icon appears in place of the green breakdot. You can clear the breakpoint by clicking the breakpoint icon.
4. A double-right click on a line of code executes a run-to-here command. This is an easy way to run to a desired point without having to set a breakpoint and then use the Go button .
5. The **e** command (short for **examine**), is useful for arbitrarily navigating through code. Enter **e** in the command pane, followed by an address or the name of a function or file. You can also use wildcards.

Chapter 3

Notes for the Pro

This Chapter Contains:

- The MULTI Builder
- Inheritance and Setting Options in the Builder
- Start MULTI with Top-Level .bld Files Only
- Specific Processor Selection
- Show Progress
- Link Control Files
- Link Maps

This Chapter provides information for people that are familiar with the basics of using MULTI.

The MULTI Builder

A graphical project hierarchy is a very powerful and intuitive concept. The MULTI project hierarchy is created by nesting **.bld** files. Each **.bld** file can contain zero or more references to other **.bld** files. In other words, you can start with one **.bld** file (typically **default.bld**) and add **.bld** files, and then **.bld** files to those, and so on in order to build a hierarchy. The **.bld** files are files on your disk.

The **.bld** files store the build/compile options that you may set for your programs/files. The MULTI Builder provides you with a graphical interface for setting and changing these build/compile options.

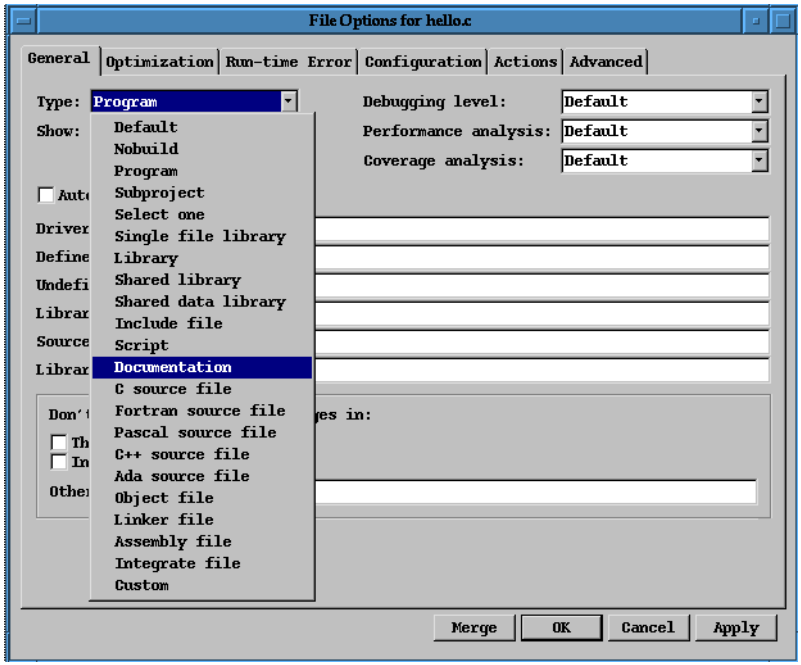
The **.bld** files are not makefiles. The MULTI Builder and its **.bld** files provide you with a graphical *alternative* to using makefiles. If you want or need to use makefiles, do not use the MULTI Builder. Since you can call the GHS compilers from the command line and consequently a makefile, you have all of the capabilities of any traditional compiler *and* a graphical build environment.

Building can be accomplished from the command line just like Make. Use **build.exe** to begin a build. For example, you can enter:

```
% build default.bld
```

Build is useful when you want to use a graphical interface to intuitively construct your program, but you want to initiate the actual build from the command line.

Each **.bld** file will be of some type, like `[nobuild]`, `[program]`, `[library]`, etc. You can set the **.bld** file's type by highlighting it and then setting in the Builder's menu bar: **Project > Options for *current file***.



By doing this, you're telling MULTI what you want to build. For example, if you have source files contained within a **hello.bld** `[program]`, you're telling MULTI to compile the contained source files and to link them into a program with the name **hello**. If you change **hello.bld** to be type `[library]`, then MULTI compiles the contained source files and then combines them into a library with the name **hello.a** (the extension varies with the target processor). **default.bld** is normally left as type `[nobuild]` which is considered a project.

Inheritance and Setting Options in the Builder

You can set various build and/or compile-time options from the **Project** menu. When you set an option, that option is set (see below for the three settings available) for the file that you highlighted in the Builder's Project pane. If the highlighted file happens to be a **.bld** file that contains other files within it, the option you set is inherited by the files contained within that **.bld** file.

Many build/compile-time options appear as a check box and have three possible settings. You can set an option to **ON**, **OFF**, or **INHERIT**.

ON Forces the option to **ON**, regardless of the setting for any higher-level (parent) **.bld** file containing the current file.

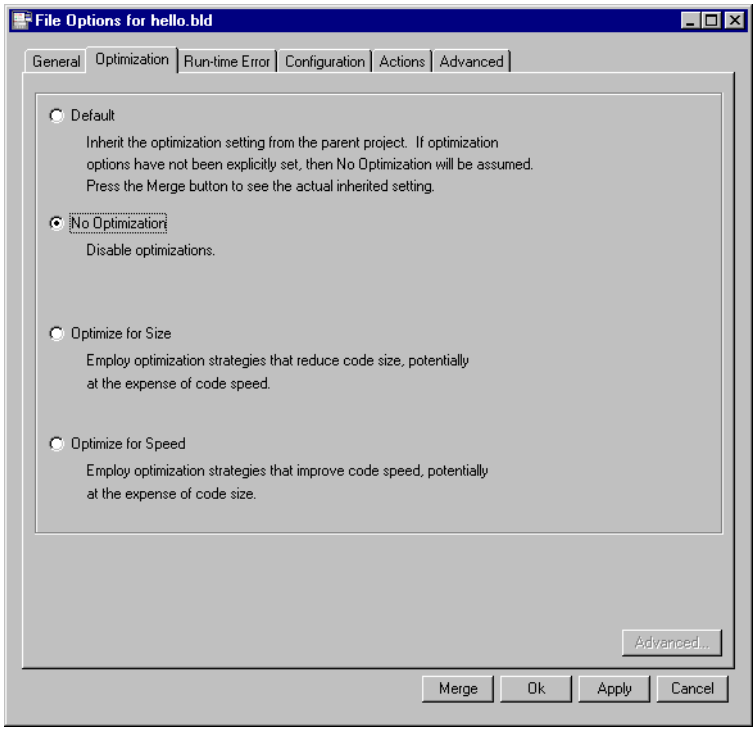
OFF Forces the option to **OFF**, regardless of the setting of any parent **.bld** file.

INHERIT Sets the option the same as it is set in the parent **.bld** file.

1. To set a build/compile-time option **ON** for a file, highlight the file of interest, then click **Project > xxxxx Options for *current file*** in the Builder's menu bar. Click the desired field until it shows a plus sign (+).
2. To set a build/compile-time option **OFF** for a file, highlight that file and click **Project > xxxxx Options for *current file*** in the Builder's menu bar. Click the desired field until it shows a minus sign (-).
3. To set a build/compile-time option to **INHERIT**, highlight that file and click **Project > xxxxx Options for *current file*** in the Builder's menu bar. Click the desired field until the field is blank.

When an option shows **Blank**/**Default**, then you may wonder, "just what is inherited for that specific file?" By clicking the Merge button  in the Builder window, the **Options** window shows what options are actually going to be applied to a specific file. Notice that the fields are grayed out so you cannot edit them. When you click the Unmerge button  , the fields become editable again.

For example, if you set in the Builder's menu bar **Project> Options for *current file* > No Optimization** on your top-level project **default.bld**, all programs and their source files contained in your **default.bld** project is compiled with **No Optimization**. Alternately, you could have set this option on just one file (i.e. **hello.c**) and left all other programs and/or source files alone.



Start MULTI with Top-Level .bld Files Only

If you understand MULTI **.bld** files, the hierarchy, and the inheritance model, you'll notice that options set at a particular level are stored at that level.

Having separate **.bld** files not only lends itself to a clean and concise model that follows a natural hierarchy that often exists, but it allows flexibility.

Sometimes your situation will call for two projects (A and B) that share a common subproject (C). One pitfall exists here: You must not start MULTI directly on the shared subproject (i.e. **multi c.bld**). If you do, MULTI has no way of knowing where to find the inherited options (A or B?).

You may not have shared subprojects, but this points out that it is important to consistently start MULTI with a top-level project as opposed to a lower-level

project. “Jumping into the middle of a hierarchy” occurs if you don’t start with a top-level project.

Specific Processor Selection

During this introduction to MULTI, you may have wondered exactly which target processor was used. The specific processor selection can be made through the Builder’s menu bar (**Project > CPU Options for *current file***) and is typically done at your top level project (i.e. **default.bld**).

Processor selection is important. If it is incorrect or left to some unknown default, your code may not run correctly on your particular target.

Processor selection not only affects compiler/code generation, but running an Instruction Set Simulator also needs to be taken into account. The Instruction Set Simulators require a parameter (omitting this parameter selects the default processor for that particular architecture) that specifies exactly which processor to simulate. The syntax is:

simulator_name [*cpu*]

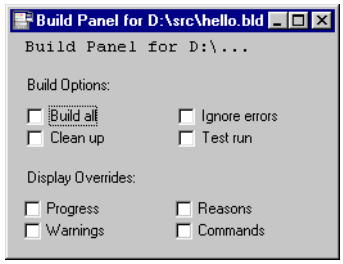
For example, **simarm -arm9e** instructs the MCore Instruction Set Simulator specifically to simulate the ARM9E. For the processors available for your product, please consult the Green Hill’s Development Guide for your target.

Show Progress

When you start a build, MULTI provides a separate Progress window. From this window, you can do many things like launch an Editor by double-clicking a syntax error message, or save the results to a file (since the Progress window is an Editor after all). In default mode, the Progress window displays simple messages showing its progress (i.e. `Compiling`…).

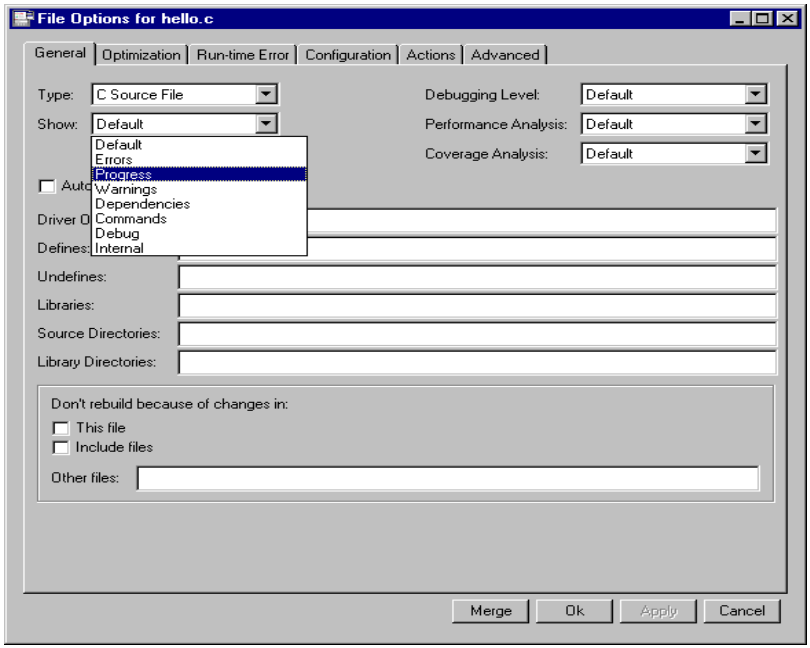
If you want a more verbose progress listing, you can either:

1. Choose from the Builder's menu bar, **Build > Advanced Build Controls...** . This control is per MULTI session.



2. In the Builder's menu bar, set **Project > Options for *current file* > Show > Progress**. This option, like any other build/compile-time option, is set on the currently highlighted file (and is inherited of course)

.



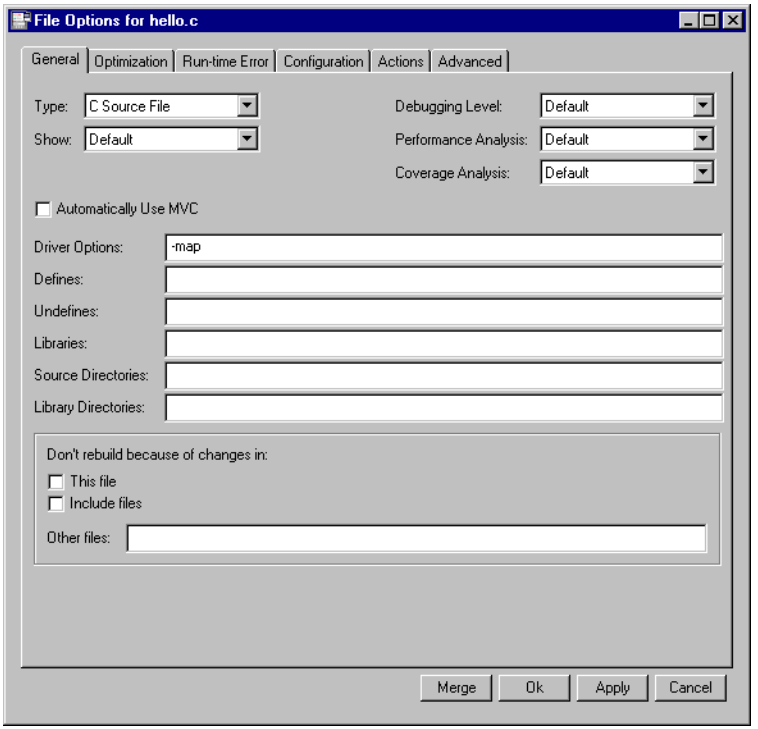
Showing a verbose progress listing is helpful when you're curious about which libraries and directories the Builder is using during compilation.

Link Control Files

To specify how your program is linked and located in memory, you can use a link control file. These files typically have a **.lnk** extension. From within the Builder, you can add this file to your `[program]`, and it is used during link. If you do not specify a **.lnk** file, the linker uses the **default.lnk** link control file from the appropriate library directory.

Link Maps

You can view a link map with the Green Hills compiler/tool chain using the **-map** option. You can use **Project>Options for *current file*>DriverOptions** field in the Builder's menu bar to specify certain options like **-map**.



Warning: The **Driver Options** field is a catch-all field that is used only when a graphical check box or field does not currently exist. Using switches that duplicate or conflict with existing graphical settings will result in indeterminate behavior. The **Driver Options** field requires that entries be separated by a comma with no spaces between them.

Handy note: You can add a *program***.map** entry to your project. This does not necessarily tell MULTI to create a link map, but provides an easy way to launch an Editor on it (i.e. double-click it).