

Getting Started with MULTI[®]
for Windows,
PowerPC



Version 1.8.9

Copyright © 1999 by Green Hills Software, Inc. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission from Green Hills Software, Inc.

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.

Green Hills Software and the Green Hills logo are trademarks, and MULTI is a registered trademark, of Green Hills Software, Inc. Microsoft is a registered trademark, and Windows, Windows 95, Windows 98, and Windows NT are trademarks of Microsoft Corporation.

All other trademarks or registered trademarks are property of their respective companies.

Revision History

| Revision | Release Date | Location of Revision(s) |
|----------|--------------|-------------------------|
| | | |
| | | |
| | | |

PubID: Q56W-C0599-89NG

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" 3

So you want to start using MULTI! We've got just the thing. This introduction gives you all the info you need to get up and running with MULTI in just 10 quick steps. This introduction quickly guides you through creating the quintessential introductory program, "Hello World!".

CHAPTER 2, "GET THE DETAILS" 7

So, now you have some time and you want to get past the surface level on MULTI. This chapter takes you through the same steps as the Up and Running... chapter but this time gives you the meat and potatoes of what's going on.

CHAPTER 3, "NOTES FOR THE PRO" 19

OK, you've completed the tutorials and now you're hungry for more. Or you are already familiar with MULTI and want to learn a bit more. This Chapter is for you. It goes just beyond the stuff in the tutorials, including information on inheritance, processor selection and link files.

Chapter

1

**UP AND RUNNING IN 10
QUICK STEPS**

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 *MULTI Reference 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 7 provides a more in depth version of these steps. Notes for the Pro on page 19 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. Using a DOS window or the MS Windows Explorer, create a temporary directory called `hello` and `cd` to it.

```
C:\>mkdir hello
C:\>cd hello
```

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.: `C:\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.

```
C:\hello>multi
```

3. ADD A [PROGRAM] OBJECT TO DEFAULT.BLD

To add a [program] object, type `hello.bld` in the field and press the button or press **Enter**.

4. NAVIGATE DOWN TO HELLO.BLD

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

5. ADD A C SOURCE FILE TO HELLO.BLD

To add a C source file to `hello.bld`, type `hello.c` in the field and press the button or press **Enter**.


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&Quit**) 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 GHS SIMULATOR

Type **simppc** into the  field. A listing of simulator and server names is in Table 2 on page 13. Press the  button or press **Enter**. Two small windows (the Target and I/O window) will appear upon successful completion. If you are using a different target board, see Connect to a GHS simulator on page 12.

9. START THE DEBUGGER

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

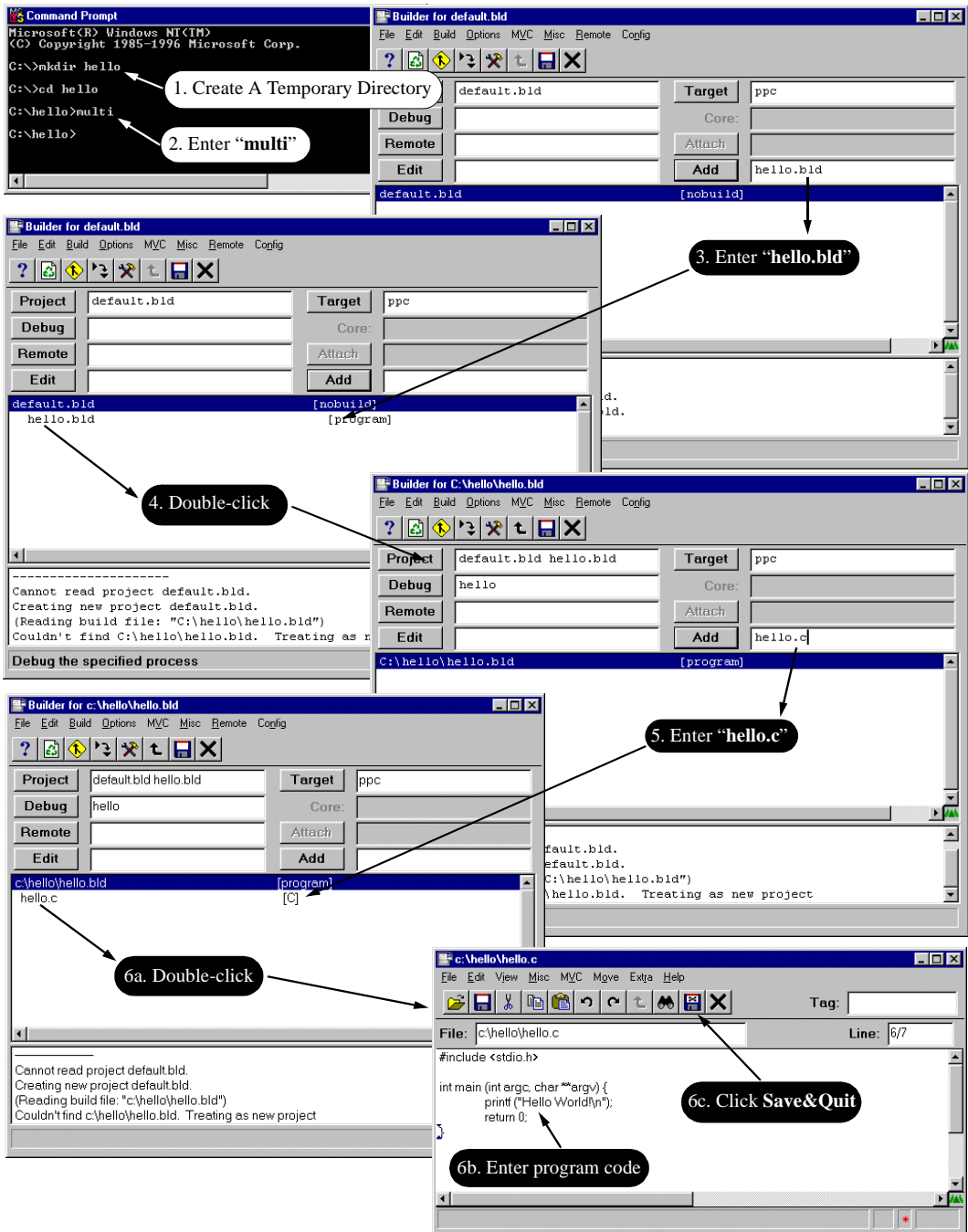
With these 10 simple steps, you have created a MULTI project, written, compiled, and run a program, and produced program output on the host. Progressing to larger applications with multiple program elements is easy. Each element is built and run the same way. Consult the *MULTI Reference Manual* for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

Introduction to MULTI® 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.



Builder for C:\hello\hello.bld

File Edit Build Options MYC Misc Remote Config

Project: default.bld hello.bld Target: ppc

Debug: hello Core: [program]

Remote: simpcc Attach: []

Edit: [] Add: []

C:\hello\hello.bld [program]

hello.c [C]

7. Click Build

8. Enter "simpcc"

Couldn't find C:\hello\hello.
 Saved C:\hello\hello.bld
 Saved default.bld
 Connected to debug server 'C:
 Remote cpu: PowerPC 500/800

progress.001

File Edit View Misc MYC Move Extra Help

File: progress.001 Line: 6/6

Analyzing C:\hello\hello.c because "hello.inf" doesn't exist
 Compiling C:\hello\hello.c because "hello.o" doesn't exist
 Linking hello
 Translating symbols for hello
 Done
 Build completed

9. Click Debug

hello

Control Run Display Show Remote State Builder Misc Config

```

1 #include <stdio.h>
2
3 int main (int argc, char **argv) {
4     print ("Hello World!\n");
5     return 0;
6 }
7

```

10. Click Next

STOPPED file: hello.c proc: main()

running 'hello'

C:\green\simpcc

IN/OUT ?

C:\green\simpcc

TARGET ?

SIM-PowerPC%

help go step next return halt restart
 detach setargs locals profile calls upstack downstk
 stops regs search pop assem edit builder
 update quit

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 *MULTI Reference 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. Using a DOS window or the MS Windows Explorer, create a temporary directory called `hello` and `cd` to it.

```
C:\>mkdir hello
C:\>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.: `C:\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.

```
C:\hello>multi
```

- ▲ If you are on a command line, you can use either of the following methods:
 1. Enter **multi** if **multi.exe** is in your `PATH` environment variable.
 2. Specify an absolute path like `c:\green\multi`. In this case, **MULTI** will find all of its necessary sub-components in the directory in which **MULTI** resides.
- ▲ If you are using Microsoft Windows and you create an icon/shortcut, make sure the working directory points to the directory where you want your **.bld** files to go. You can do this by right-clicking the shortcut, and then setting **Properties->Shortcut->Start in** to the desired directory. This is a standard Microsoft Windows convention.
- ▲ 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, enter **hello.bld** in the field.

- ▲ You may refer to *The MULTI Builder* on page 20 for information about the Builder.
- ▲ In this step, you're basically creating one program called **hello** in your project, **default.bld**.

4. NAVIGATE DOWN INTO HELLO.BLD


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

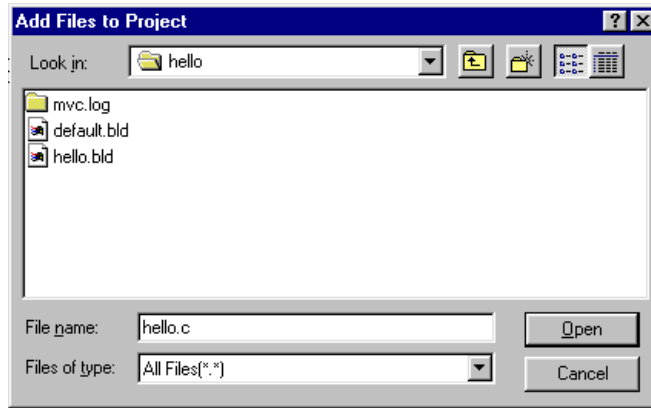
When you navigate to a **.bld** file, you are essentially opening that **.bld** file like an Editor opens a text file. The **.bld** file currently open appears at the top of the Project pane. When a build/compile option is set, that option is stored in the open **.bld** file.

5. ADD A C SOURCE FILE TO HELLO.BLD

To add a C source file to **hello.bld**, type **hello.c** in the field and press the button or press **Enter**.

- ▲ 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.


- ▲ If the **Add** field is empty and you click the  button, an **Add File** dialog appears.



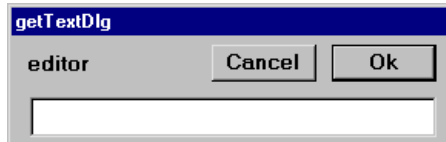
This allows you to browse for desired files. You can select multiple files in the **Add Files** dialog by holding down the <Ctrl> key while selecting files.

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&Quit**) button to save the file and exit the Editor.

- ▲ MULTI can launch a different editor than the integrated MULTI Editor. In the menu bar in the Builder, go to **Config->Strings->Editor**, and enter the location of your editor.





Refer to the *MULTI Reference Manual* 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.

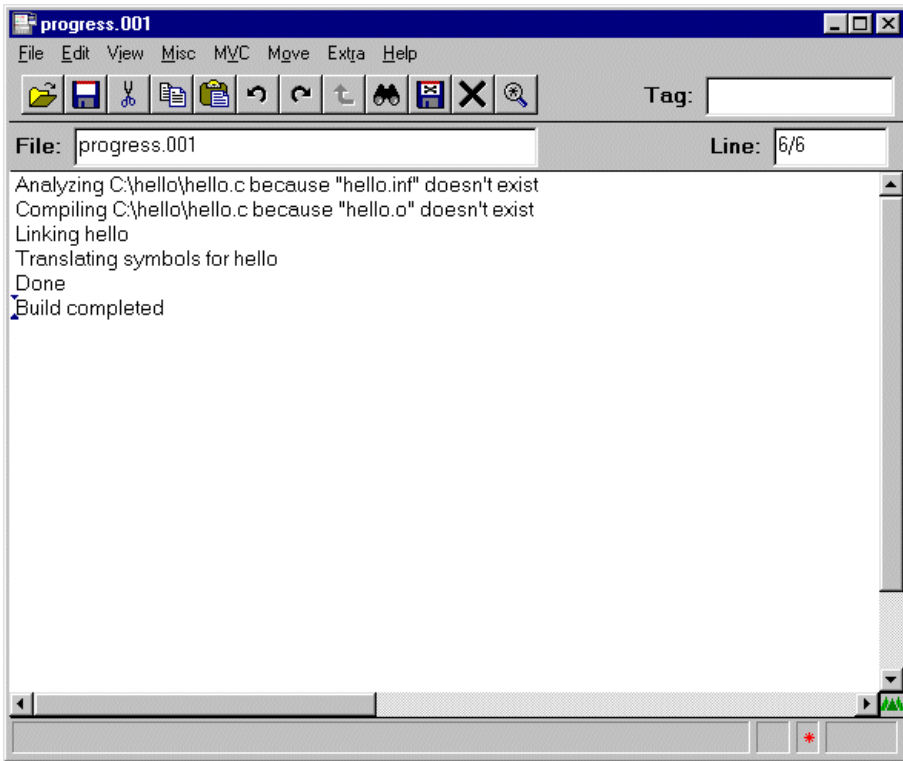
7. COMPILE BY CLICKING THE BUILD BUTTON IN THE BUILDER

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** in the Builder's menu bar. **Build->Clean** and **BuildAll** 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 **X** (Exit) button.

8. CONNECT TO A GHS SIMULATOR

Type **simpcc** into the **Remote** field. A listing of simulator and server names is in Table 2 on page 2-13. Press the **Remote** button or press **Enter**. Two small windows (the Target and I/O window) will appear upon successful completion. If you are using a different target board, see Debug Servers on page 13.

- ▲ **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.

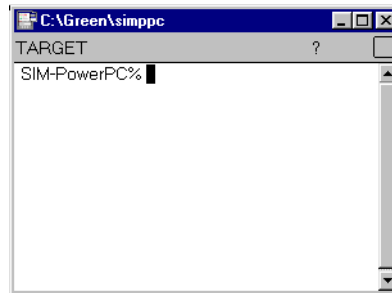
- ▲ Pre-build demos for use with several boards are provided with your distribution. These demos are complete with setup files and link files. For more information on these demos look at the readme.txt file located in C:\green\demo.
- ▲ In order to use many of the commands below you will need to copy the appropriate setup.ocd or setup.dbs file from c:\green\demo\- ▲ If you have a target board, follow the instructions included with the board's documentation for properly installing the board. Connect to the board by entering the remote command from the following table in the field. 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 Boards | Command |
|-----------------|--|--|
| PPC Simulator | | simppc |
| Network OCD | ADS821, MBX821, ADS860, MBX860 | ocdserv tcp <i>Targetname</i> ppc800 -s setup.ocd |
| Hummingbird OCD | ADS821, MBX821, ADS860, MBX860 | ocdserv hblpt1 ppc800 -s setup.ocd |
| MaCraigor OCD | ADS821, MBX821, ADS860, MBX860, EVB555, IBM403, Excimer | ocdserv lpt1 [ppc800 ppc403 ppc555 ppc603rr] -s setup.ocd |
| Raven OCD | Excimer | ocdserv rlpt1 -s setup.ocd |
| EST ICE | ADS821, MBX821, ADS860, MBX860, EST800 | estserv tcp <i>Targetname</i> ppc860 -s setup.ocd |
| HP ICE | ADS821, MBX821, ADS860, MBX860, EVB555, Excimer, VADS8260, IBM403 | hpserv -s setup.dbs -nobss <i>Targetname</i> |
| Dink Monitor | Excimer | dinkserv com1 |
| Tornado | MV1603, MV1604, MV2603, MV2604 | tornserv -vio <i>Targetname</i> |

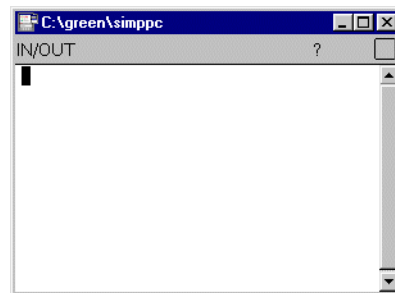
Table 1 Supported PPC Debug Servers

- ▲ 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.



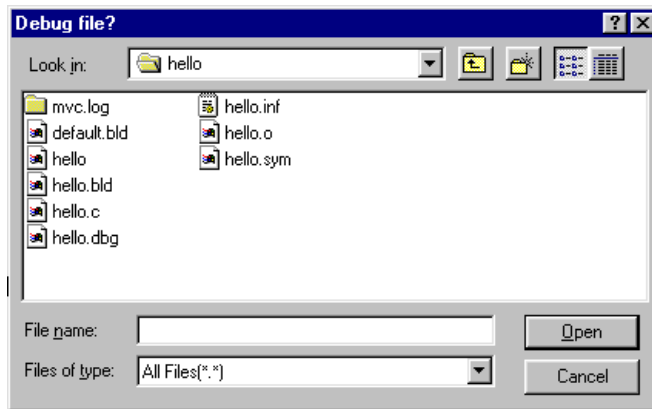
- ▲ For this example, you used an instruction set simulator. The GHS instruction set simulators are intended to provide instruction set valid simulation. Some of them provide limited forms of cycle accurate and cache simulation. However, extensive hardware simulation (including on-chip and on-board peripherals) 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 simulator (or downloads it to the remote

target, depending on what you entered in Step 8) and opens the MULTI Debugger window with the program source code displayed.



- ▲ The **Debug** field is filled in for you with the name of the program that you have opened in the Builder. You only need to click the **Debug** button.
- ▲ If the **Debug** field is empty and you click the **Debug** button, a **Debug** file dialog allows you to browse for an executable.



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 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 *MULTI Reference 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 **assem** button is useful for displaying source code and assembly interlaced. The **step** and **next** buttons 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. Use the **pop** button to navigate backwards through the path you may have taken.

NOTES FOR THE PRO

This Chapter Contains:

- ▲ More information on the MULTI Builder
- ▲ Setting options
- ▲ Useful information for using MULTI

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.

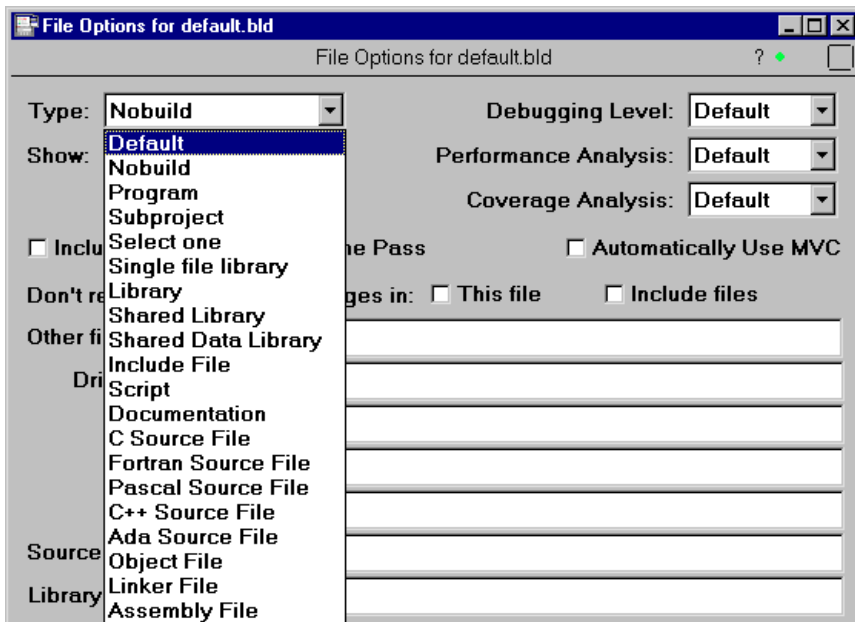
The **.bld** files are not makefiles. The MULTI Builder and its **.bld** files provide you with a graphical *alternative* to using makefiles. If you want or need to use makefiles, do not use the MULTI Builder. Since you can call the GHS compilers from the command line and consequently a makefile, you 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:

```
C:\hello>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: **Options->FileOptions...->Type**.



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 **Options** 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**.

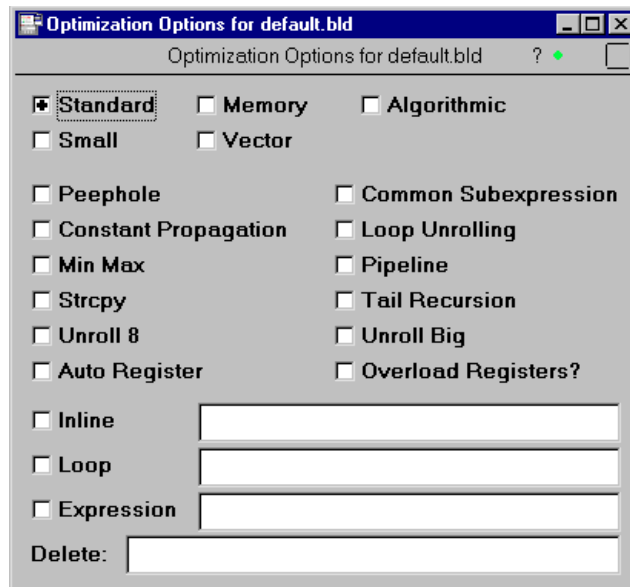
- ON** Forces the option to **ON**, regardless of the setting for any higher-level (parent) **.bld** file containing the current file.
- OFF** Forces the option to **OFF**, regardless of the setting of any parent **.bld** file.
- INHERIT** Sets the option the same as it is set in the parent **.bld** file.

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

Note: When you set an option, it is applied immediately. There are no "Apply" or "OK" buttons in the **Options** windows.

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 **Options->Optimization...->StandardOptimization** on your top-level project **default.bld**, all programs and their source files contained in your **default.bld** project is compiled with **Standard Optimization**. Alternatively, you could have set this option on just one file (i.e. **hello.c**) and left all other programs and/or source files alone.



THE TARGET FIELD

By default, the **TARGET** field in the Builder is actually a reference to a **.bld** file. That **.bld** file contains the minimum build/compile options that need to be set in order for the Builder to build something reasonable. Since the Builder is used for a very large array of processors, targets, RTOS's, etc., the Builder is very flexible and needs a starting point. The default TARGET.bld files that come with your GHS distribution provide such starting points.

In line with the Builder inheritance model, options in the *TARGET.bld* file are inherited hierarchically downward into your **default.bld**, and further down into **hello.bld** (unless blocked, of course).

By default, MULTI determines what the **TARGET** is by looking at the **target.def** file in the green directory. If you override the default **TARGET** (by clicking the **Target** button and selecting a different *TARGET.bld* file from the green directory), that selection is used and stored in your **default.bld** (just as other build/compile options are stored in a **.bld** file).

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.

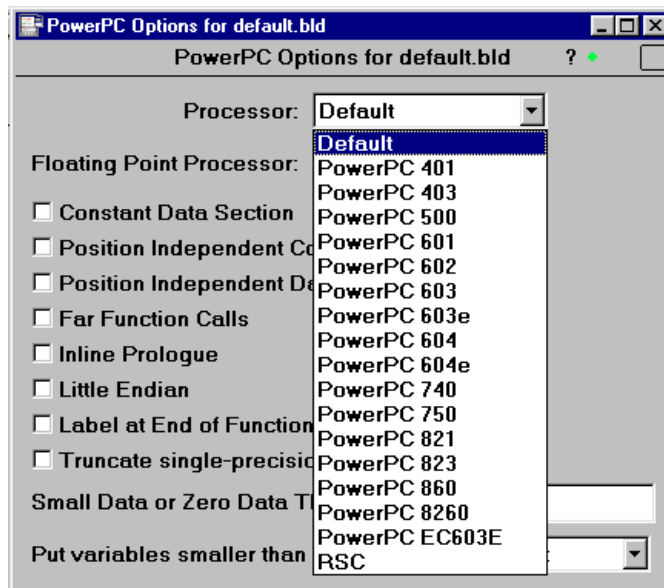
Having separate **.bld** files not only lends itself to a clean and concise model that follows a natural hierarchy that often exists, but it allows flexibility.

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 (**Options->CPU...->Processor**) and it 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, **simppc 68020** instructs the PowerPC Instruction Set Simulator specifically to simulate the PPC860. For the processors available for your product, please consult the Green Hill's Development Guide for your target.

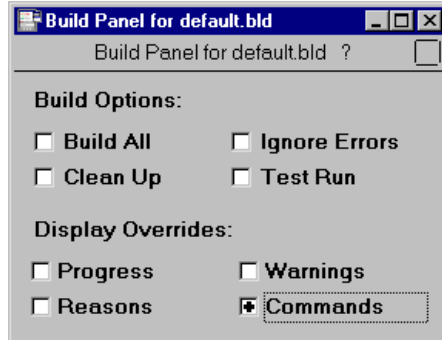
SHOW PROGRESS

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

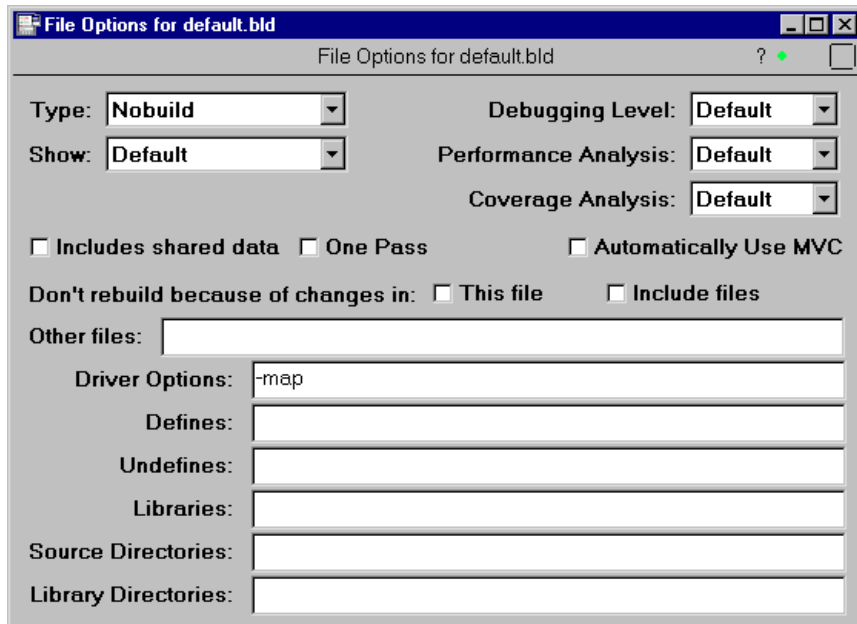
an Editor after all). In default mode, the Progress window displays simple messages showing its progress (i.e. Compiling...).

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

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



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



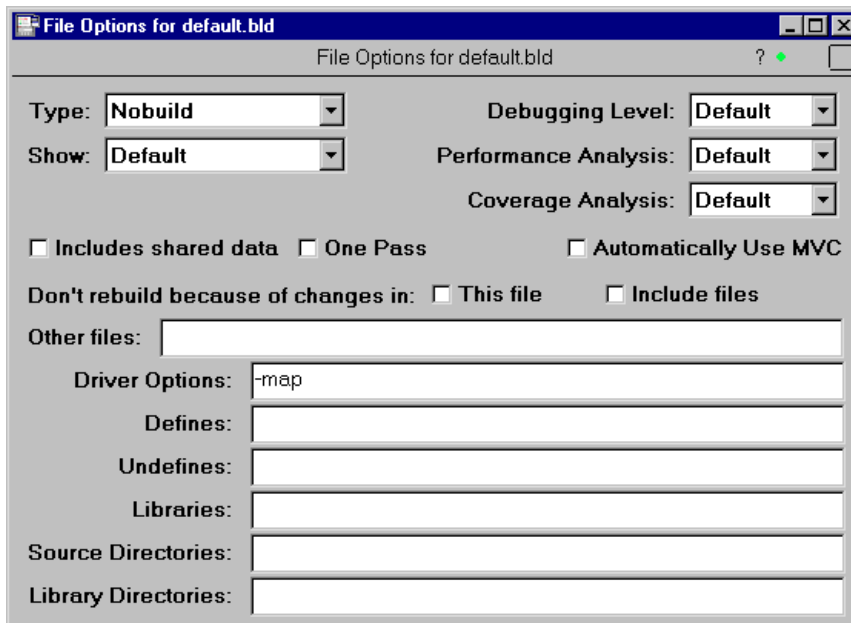
Showing a verbose progress listing is helpful when you're curious about which libraries and directories the Builder is using during compilation.

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 **Options->FileOptions->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. Additionally, 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).