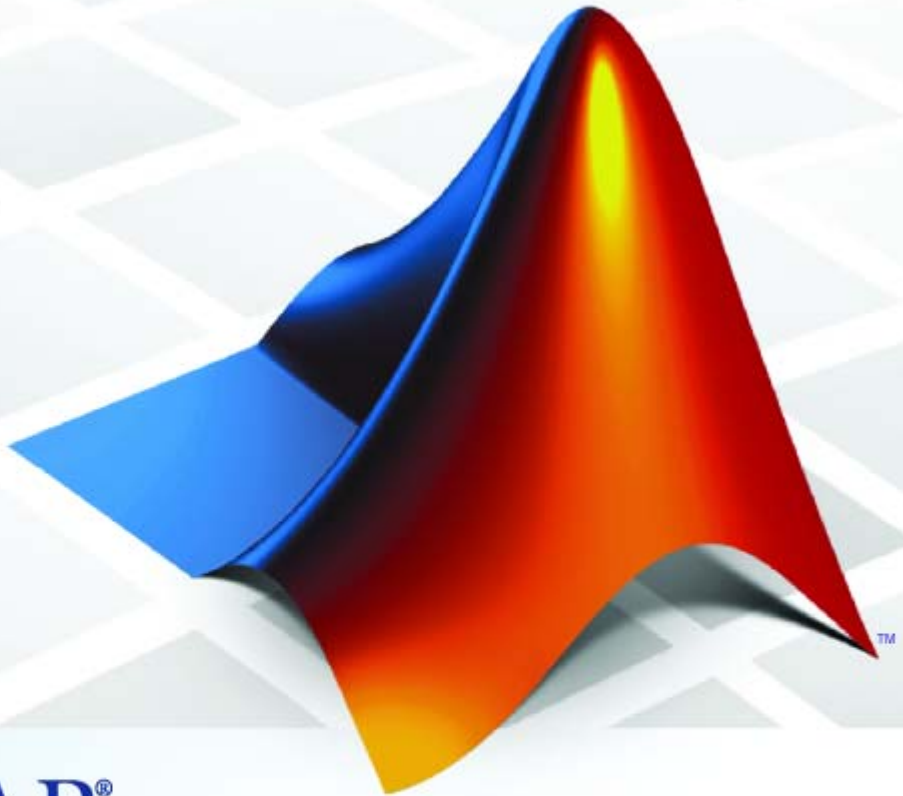


# Embedded IDE Link™ 4 User's Guide

*For Use with Green Hills® MULTI®*



**MATLAB®**

## How to Contact The MathWorks



[www.mathworks.com](http://www.mathworks.com) Web  
[comp.soft-sys.matlab](mailto:comp.soft-sys.matlab) Newsgroup  
[www.mathworks.com/contact\\_TS.html](http://www.mathworks.com/contact_TS.html) Technical Support



[suggest@mathworks.com](mailto:suggest@mathworks.com) Product enhancement suggestions  
[bugs@mathworks.com](mailto:bugs@mathworks.com) Bug reports  
[doc@mathworks.com](mailto:doc@mathworks.com) Documentation error reports  
[service@mathworks.com](mailto:service@mathworks.com) Order status, license renewals, passcodes  
[info@mathworks.com](mailto:info@mathworks.com) Sales, pricing, and general information



508-647-7000 (Phone)



508-647-7001 (Fax)



The MathWorks, Inc.  
3 Apple Hill Drive  
Natick, MA 01760-2098

For contact information about worldwide offices, see the MathWorks Web site.

*Embedded IDE Link™ User's Guide*

© COPYRIGHT 2007-2010 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

### Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See [www.mathworks.com/trademarks](http://www.mathworks.com/trademarks) for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

### Patents

The MathWorks products are protected by one or more U.S. patents. Please see [www.mathworks.com/patents](http://www.mathworks.com/patents) for more information.

**Revision History**

November 2007 Online only  
March 2008 Online only  
October 2008 Online only  
March 2009 Online only  
September 2009 Online only  
March 2010 Online only

New for Version 1.0 (Release 2007b+)  
Revised for Version 1.0.1 (Release 2008a)  
Revised for Version 1.1 (Release 2008b)  
Revised for Version 1.2 (Release 2009a)  
Revised for Version 4.0 (Release 2009b)  
Revised for Version 4.1 (Release 2010a)



## Getting Started

### 1

<b>Product Overview</b> .....	1-2
<b>Software Structure and Components</b> .....	1-4
Components .....	1-4
Automation Interface .....	1-4
Project Generator .....	1-5
Verification .....	1-5
Configuring Your Software .....	1-6
Configuring Green Hills® MULTI to use Full Directory Paths .....	1-9
<b>Software Requirements</b> .....	1-10

## Automation Interface

### 2

<b>Getting Started with Automation Interface</b> .....	2-2
Introducing the Automation Interface Tutorial .....	2-2
Starting and Stopping Green Hills MULTI From the MATLAB Desktop .....	2-4
Running the Interactive Tutorial .....	2-8
Querying Objects for Green Hills MULTI Software .....	2-8
Loading Files into Green Hills MULTI Software .....	2-9
Running the Project .....	2-11
Working With Data in Memory .....	2-12
More Memory Data Manipulation .....	2-14
Closing the Connections to Green Hills MULTI Software ..	2-17
Tasks Performed During the Tutorial .....	2-17
<b>Constructing Objects</b> .....	2-19
Example — Constructor for ghsmulti Objects .....	2-19

<b>Properties and Property Values</b> .....	2-21
Working with Properties .....	2-21
Setting and Retrieving Property Values .....	2-21
Setting Property Values Directly at Construction .....	2-22
Setting Property Values with set .....	2-22
Retrieving Properties with get .....	2-23
Direct Property Referencing to Set and Get Values .....	2-23
Overloaded Functions for ghsmulti Objects .....	2-24
<b>ghsmulti Object Properties</b> .....	2-25
Quick Reference to ghsmulti Properties .....	2-25
Details About ghsmulti Object Properties .....	2-25

## Project Generator

### 3

<b>Introducing Project Generator</b> .....	3-2
<b>Project Generator Tutorial</b> .....	3-3
Process for Building and Generating a Project .....	3-3
Create the Model .....	3-4
Adding the Target Preferences Block to Your Model .....	3-5
Specifying Simulink Configuration Parameters for Your Model .....	3-7
Creating Your Project .....	3-9
<b>Model Reference</b> .....	3-11
About Model Reference .....	3-11
How Model Reference Works .....	3-11
Using Model Reference .....	3-12
Configuring Targets to Use Model Reference .....	3-14

## Block Reference

### 4

<b>Block Library: idelinklib_ghsmulti</b> .....	4-2
---	-----







# Getting Started

---

- “Product Overview” on page 1-2
- “Software Structure and Components” on page 1-4
- “Software Requirements” on page 1-10

## Product Overview

Embedded IDE Link™ software provides an interface between MATLAB® and the Green Hills MULTI® IDE software. The software enables you to

- Access the processor
- Manipulate data on the processor
- Manage projects within the IDE

while using the MATLAB numerical analysis and simulation functions.

Embedded IDE Link software connects MATLAB and Simulink® with Green Hills MULTI integrated development and debugging environment from Green Hills®. The software enables you to use MATLAB and Simulink to debug and verify embedded code running on many microprocessors that Green Hills MULTI software supports, such as the ARM®, Freescale™ MPC5500 and MPC7400, Blackfin®, and NEC® V850 families.

Using the software, you can perform the following tasks and others related to Model-Based Design:

- Function calls — Write scripts in MATLAB to execute any function in the Green Hills MULTI IDE
- Automation — Write automated tests in MATLAB to execute on your processor, including control and verification operations
- Host-Processor Communication — Communicate with the processor directly from MATLAB, without going to the IDE
- Verification and Validation
  - Load and execute projects into the Green Hills MULTI IDE software from the MATLAB command line
  - Build and compile code, and then use vectors of test data and parameters to test the code
  - Build and compile your code, and then download the code to the processor and execute it

- Design models — Design models and algorithms in MATLAB and Simulink and run them on the processor
- Generate code — Generate executable code for your processor directly from the models designed in Simulink, and execute it

Embedded IDE Link software includes a project generator component. With the project generator component, you can generate a complete project file for Green Hills MULTI software from Simulink models, using C code generated with Real-Time Workshop<sup>®</sup> software. Thus, you can use both Real-Time Workshop and Real-Time Workshop<sup>®</sup> Embedded Coder<sup>™</sup> software to generate generic ANSI C code projects for Green Hills MULTI from Simulink models. You can then build and run the code on supported processors.

The following list suggests some of the uses for Embedded IDE Link software:

- Create test benches in MATLAB and Simulink for testing your manually written or automatically generated code running on a variety of DSPs
- Generate code and project files for Green Hills MULTI software from Simulink models using both Real-Time Workshop and Real-Time Workshop Embedded Coder software for rapid prototyping or deployment of a system or application
- Build, debug, and verify embedded code on supported processors with MATLAB, Simulink, and Green Hills MULTI software
- Perform processor-in-the-loop (PIL) testing of embedded code

## Software Structure and Components

### In this section...

“Components” on page 1-4

“Automation Interface” on page 1-4

“Project Generator” on page 1-5

“Verification” on page 1-5

“Configuring Your Software” on page 1-6

“Configuring Green Hills® MULTI to use Full Directory Paths” on page 1-9

### Components

Embedded IDE Link software comprises these components

- Automation Interface — Enables communication between MATLAB and Green Hills® MULTI® software.
- Project Generation — Uses Simulink to let you build models, simulate them, and generate code from the models directly to the processor.
- Verification — Validate and verify your projects. You can simulate algorithms and processes in Simulink models and concurrently on your processor. Comparing the concurrent simulation results helps verify the fidelity of your model or algorithm code.

### Automation Interface

The Automation Interface component enables you to use MATLAB functions and methods to communicate with the Green Hills MULTI IDE software. With the MATLAB functions, you can perform the following program development tasks:

- Automate project management.
- Debug projects by manipulating the data in the processor memory (internal and external) and registers.
- Exercise functions from your project on the processor.

- Communicate between the host and processor applications.

The Automation Interface component provides the following functionality in the Debug component—methods and functions for project automation, debugging, and data manipulation.

## Project Generator

The Project Generator component is a collection of methods that use the Green Hills MULTI API to create projects in Green Hills MULTI and generate code with Real-Time Workshop. With the interface, you can do the following:

- Automatic project-based build process — Automatically create and build projects for code generated by Real-Time Workshop or Real-Time Workshop Embedded Coder.
- Custom code generation — Use Embedded IDE Link software with any Real-Time Workshop Embedded Coder System Target File (STF) to generate both processor-specific and optimized code.
- Automatic downloading and debugging — Debug generated code in the Green Hills MULTI debugger, using either the instruction set simulator or real hardware.
- Create and build projects for Green Hills MULTI from Simulink models — Project Generator uses Real-Time Workshop or Real-Time Workshop Embedded Coder to build projects that work with supported processors.
- Generate custom code using the Configuration Parameters in your model with the system target files `multilink_ert.tlc` and `multilink_grt.tlc`.

## Verification

Verifying your processes and algorithms is an essential part of developing applications. The components of Embedded IDE Link software provide the following verification tools.

- **Processor in the loop (PIL) cosimulation** — Use cosimulation techniques to verify generated code running in an instruction set simulator or real hardware environment.

- **Execution profiling** — Gather execution profiling measurements with Green Hills MULTI instruction set simulator to establish the timing requirements of your algorithm.

## Configuring Your Software

Embedded IDE Link software requires some information about your MULTI installation before you can use the software to develop projects in MULTI from MATLAB. To configure the interface between MATLAB and MULTI, provide the information in the following table. Embedded IDE Link software provides a GUI-based configuration utility to help you configure the software and interface.

GUI Parameter	Configuration Information	Description
Directory	MULTI installation directory	Identifies the path to your Green Hills software.
Configuration	Primary processor	Identifies the processor on which you are developing.
Debug server	Debug server type	Specifies the type of debug server to use.
Host name	Host name	Specifies the name of the machine that runs your Embedded IDE Link service.
Port number	Port number	Specifies the port for communicating with the host and Embedded IDE Link service. The service listens on this port.

## Configuring Embedded IDE Link Software

You must configure your installation before you start working with the software and MULTI.

To generate code for Blackfin processors, the software supports only the Green Hills version of the Blackfin compiler.

---

**Note** The software does not support using Analog Devices™ Blackfin® compiler. When you select your configuration during the configuration process, do not select `bfadi_standalone.tgt` from the **Configuration** list. `bfadi_standalone.tgt` uses the ADI compiler.

---

Follow these steps to open the Embedded IDE Link configuration utility:

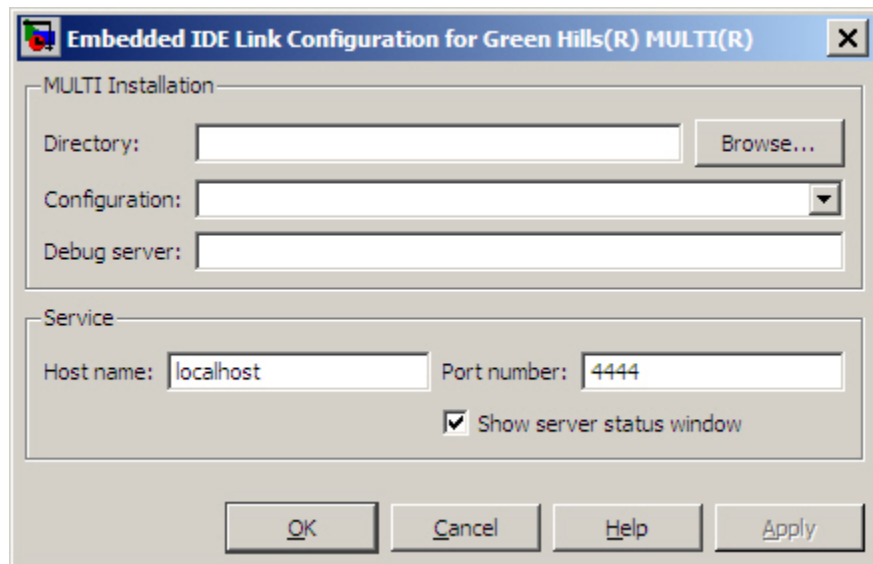
---

**Note** You must perform this configuration process before using Embedded IDE Link software.

---

- 1 Enter `ghsmulticonfig` at the MATLAB prompt.

The Embedded IDE Link Configuration dialog box opens, as shown in the following figure.



- 2** In the **Directory** field, enter the path to the executable file `multi.exe` for your Green Hills MULTI installation. Click **Browse** to search for the file if necessary.
- 3** From the **Configuration** list, select your primary processor. Embedded IDE Link software supports a variety of processors. Choose one that matches your development platform. In many cases, the `processor_standalone.tgt` variants, such as `ppc_standalone.tgt`, work well. Refer to your Green Hills MULTI documentation for more information about the configuration options for processors.
- 4** Enter the debug server string in **Debug server**. The string you enter sets specific values for processors, such as the board support library and whether the processor is big or little endian.

The standard input string is `debugconnection`. To use a processor simulator, such as the MPC5554 simulator, enter the string

```
simppc -cpu=ppc5554 -fast -dec-rom_use_entry
```

Your MULTI documentation provides more information about the debug server options and how to use them. You can find more debug server string for simulators in the reference material for `ghsmulticonfig`.

---

**Note** If you use a custom board, add the `-bsp` option to the **Debug server** string to specify your processor. For example, add `-bsp=mpc5554` if you use the MPC5554 EVB.

---

- 5** In **Host name**, enter the name of the machine that is going to run the Embedded IDE Link service. When you construct a `ghsmulti` object, the `ghsmulti` function starts the Embedded IDE Link service. To launch the service, the function needs to know where the service will run. The **Host name** string identifies that location. The default value is `localhost`, meaning the service runs on the local machine. No other input is valid.
- 6** Enter the port number for the service in **Port number**.

Port number 4444 is the default port value. To change the port used, enter a different value in this field. Verify that the port you enter is available.



If the port number you enter is not available, the Embedded IDE Link service does not start. Thus, you get an error message in MATLAB when you try to construct a `ghsmulti` object.

- 7** Select or clear **Show server status window** to specify whether the Embedded IDE Link service status appears in the task bar. The default value is to show the service status. Clearing **Show server status window** hides the status in the task bar. Select this option as a best practice. Keeping this option selected enables the software to shut down the communication services for Green Hills MULTI completely.
- 8** Click **OK** to complete the configuration process and close the dialog box.

## **Configuring Green Hills MULTI to use Full Directory Paths**

When you install MULTI to use with the software, MULTI sets the **Show Paths** option to use relative file paths. To ensure that projects and programs build correctly, configure MULTI to use full directory paths. Follow these steps to change the configuration in MULTI.

- 1** Start MULTI from your desktop.
- 2** Switch to the Project Manager tool.
- 3** Select **View > Show Paths > Full Paths**.

## **Software Requirements**

For detailed information about the software and hardware required to use Embedded IDE Link software, refer to the Embedded IDE Link system requirements areas on the MathWorks Web site:

- Requirements for Embedded IDE Link:  
[www.mathworks.com/products/ide-link/requirements.html](http://www.mathworks.com/products/ide-link/requirements.html)
- Requirements for use with Green Hills MULTI:  
[www.mathworks.com/products/ide-link/ghs-adaptor.html](http://www.mathworks.com/products/ide-link/ghs-adaptor.html)

# Automation Interface

---

- “Getting Started with Automation Interface” on page 2-2
- “Constructing Objects” on page 2-19
- “Properties and Property Values” on page 2-21
- “ghsmulti Object Properties” on page 2-25

## Getting Started with Automation Interface

In this section...
“Introducing the Automation Interface Tutorial” on page 2-2
“Starting and Stopping Green Hills MULTI From the MATLAB Desktop” on page 2-4
“Running the Interactive Tutorial” on page 2-8
“Querying Objects for Green Hills MULTI Software” on page 2-8
“Loading Files into Green Hills MULTI Software” on page 2-9
“Running the Project” on page 2-11
“Working With Data in Memory” on page 2-12
“More Memory Data Manipulation” on page 2-14
“Closing the Connections to Green Hills MULTI Software” on page 2-17
“Tasks Performed During the Tutorial” on page 2-17

### Introducing the Automation Interface Tutorial

Embedded IDE Link software provides a connection between MATLAB software and a processor in Green Hills MULTI development environment. You use MATLAB objects as a mechanism to control and manipulate a signal processing application using the computational power of MATLAB software. This approach can help you while you debug and develop your application. Another possible use for automation is creating MATLAB scripts that verify and test algorithms that run in their final implementation on your production processor.

---

**Note** Before using the functions available with the objects, you must designate a server and processor in Green Hills MULTI software. The object you create is specific to the server and processor you specify.

---

To help you start using objects in the software, Embedded IDE Link software includes a tutorial—`multilinkautoinntutorial.m`. As you work through

this tutorial, you perform the following tasks that step you through creating and using objects to interact with the Green Hills MULTI IDE:

- 1 Select your primary server and port.
- 2 Create and query objects to Green Hills MULTI IDE.
- 3 Use MATLAB to load files into Green Hills MULTI IDE.
- 4 Work with your Green Hills MULTI IDE project from MATLAB.
- 5 Close the connections you opened to Green Hills MULTI IDE.

The tutorial covers some methods and functions for the software. The following tables show functions and methods for the software. The functions do not require an object. The methods require an existing `ghsmulti` object to use as an input argument for the method.

### Functions for Working with Green Hills MULTI

The following table shows functions that do not require an object.

Function	Description
<code>ghsmulti</code>	Construct an object that refers to a Green Hills MULTI IDE instance. When you construct the object you specify the IDE instance by host and port.
<code>ghsmulticonfig</code>	Set Embedded IDE Link software preferences.

### Methods for Working with `ghsmulti` Objects in Green Hills MULTI

The following table presents some of the methods that require a `ghsmulti` object.

<b>Methods</b>	<b>Description</b>
add	Add file to project
address	Return address and page for entry in symbol table in Green Hills MULTI IDE
build	Build project in Green Hills MULTI
cd	Change working directory
connect	Connect IDE to processor
display	Display properties of object that references Green Hills MULTI IDE
halt	Terminate execution of process running on processor
isrunning	Test whether processor is executing process
load	Load built project to processor
open	Open file in project
read	Retrieve data from memory on processor
regread	Read values from processor registers
regwrite	Write data values to registers on processor
reset	Restore program counter (PC) to entry point for current program.
restart	Restore processor to program entry point
run	Execute program loaded on processor
write	Write data to memory on processor

## **Starting and Stopping Green Hills MULTI From the MATLAB Desktop**

Embedded IDE Link software provides you the ability to control MULTI software from the MATLAB command window. When you create a `ghsmulti` object, MATLAB starts the services shown in the following table to enable MATLAB to communicate with the Green Hills MULTI IDE:

Service Type for Each Port	Process Name	Description
Python Service	mpythonrun.exe	Python is a programming language the software uses to establish a connection between MATLAB and MULTI.
Python Service	svc_python.exe	Connection to IDE.
Python Service	svc_router.exe	Connection to IDE.
Python Service	svc_statemgr.exe	Connection to IDE
Python Service	svc_window.exe	Connection to IDE.
Embedded IDE Link service	Not applicable	Enables MATLAB to send commands to the Green Hills MULTI development environment. This is a child process of the python services.

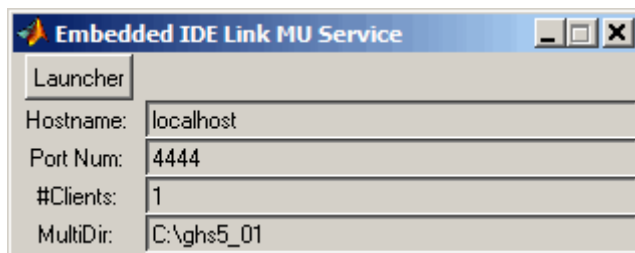
Each time you create a `ghsmulti` object, the software starts another set of the python services shown in the table.

### Starting Green Hills MULTI From MATLAB

When you use the `ghsmulti` function, the software starts two classes of services—python services and the Embedded IDE Link service for each new port. The entries in the following table describe how the software controls the IDE when you create a `ghsmulti` object:

Create ghsmulti Object with ghsmulti Function	Status of IDE	Result
id=ghsmulti	Not running	The software starts the Embedded IDE Link service and the IDE connects to the default host name and port number—localhost and 4444 as set in the configuration options.
id=ghsmulti('hostname','localhost','portnum',4444)	Not running	The software starts the Embedded IDE Link service and the IDE and connects to the specified host name and port number—localhost and 4444.
id2=ghsmulti	Running	The software connects to the existing Embedded IDE Link service connected to the default host name and port.
id2=ghsmulti('hostname','localhost','portnum',4446)	Running	The software starts a new the Embedded IDE Link service connected to the specified host name and port number.

When the software starts the Embedded IDE Link service, the following service dialog box appears on your desktop:



Information in the window provides details about the service. Clicking **Launcher** opens the MULTI Launcher utility.



## Stopping Green Hills MULTI From MATLAB

After you complete your development work with the software, best practice suggests that you close the IDE from MATLAB. Two conditions control how you close the IDE, as shown in the following table:

The Embedded IDE Link Service State	To Close the IDE
<p>One or more services appear in the task bar and the Embedded IDE Link service dialog boxes are visible.</p>	<p>Perform these steps:</p> <ol style="list-style-type: none"> <li><b>1</b> Enter <code>clear all</code> in MATLAB to remove the <code>ghsmulti</code> objects from your workspace.</li> <li><b>2</b> Verify that the MULTI clients are no longer connected by checking that <b>#Clients</b> in each service dialog box is 0.</li> <li><b>3</b> Close the service dialog boxes.</li> </ol>
<p>Services appear in the task bar but the service dialog boxes are not visible.</p>	<p>Perform these steps:</p> <ol style="list-style-type: none"> <li><b>1</b> Enter <code>clear all</code> in MATLAB to remove the <code>ghsmulti</code> objects from your workspace.</li> <li><b>2</b> Open the Microsoft® Windows Task Manager.</li> <li><b>3</b> Click <b>Processes</b>.</li> <li><b>4</b> Select <code>svc_router.exe</code> from the list. Closing this service stops <code>mpythonrun.exe</code>, <code>svc_window.exe</code>, and <code>svc_statemgr.exe</code>.</li> <li><b>5</b> Click <b>End Now</b>.</li> <li><b>6</b> Select <code>svc_python.exe</code> from the list.</li> </ol>

<b>The Embedded IDE Link Service State</b>	<b>To Close the IDE</b>
	<b>7</b> Click <b>End Now</b> .

---

**Note** Clicking the task bar icon for the service and selecting close does not close the IDE correctly.

---

## Running the Interactive Tutorial

You have the option of running this tutorial from the MATLAB command line or entering the functions as described in the following tutorial sections.

To run the tutorial in MATLAB, click run `multilinkautointtutorial`. This command launches the tutorial in an interactive mode where the tutorial program provides prompts and text descriptions to which you respond to move to the next section. The interactive tutorial covers the same information provided by the following tutorial sections. You can view the tutorial MATLAB file used here by clicking `multilinkautointtutorial.m`.

## Querying Objects for Green Hills MULTI Software

In this tutorial section you create the connection between MATLAB and Green Hills MULTI IDE. This connection, or `ghsmulti` object, is a MATLAB object that you save as variable `id`. You use function `ghsmulti` to create `ghsmulti` objects. `ghsmulti` supports input arguments that let you specify values for `ghsmulti` object properties, such as the global timeout. Refer to the `ghsmulti` reference information for more about the input arguments.

Use the generated object `id` to direct actions to your project and processor. In the following tasks, `id` appears in all method syntax that interact with the IDE primary target and the processor: The object `id` identifies and refers to a specific instance of the IDE.

You must include the object in any method syntax you use to access and manipulate a project or files in a session in Green Hills MULTI software:

- 1 Create an object that refers to your selected service and port. Enter the following command at the prompt.

```
id = ghsmulti('hostname','localhost','portnum',4444)
```

- 2 Next, enter `display(id)` at the prompt to see the status information.

```

MULTI Object:
  Host Name      : localhost
  Port Num       : 4444
  Default timeout : 10.00 secs
  MULTI Dir      : C:\ghs\multi500\ppc\

```

Embedded IDE Link software provides three methods to read the status of a processor:

- `info` — Return a structure of testable session conditions.
- `display` — Print information about the session and processor.
- `isrunning` — Return the state (running or halted) of the processor.

- 3 Verify that the processor is running by entering

```
runstatus = isrunning(id)
```

The MATLAB prompt responds with message that indicates the processor is stopped:

```
runstatus =
```

```
0
```

## Loading Files into Green Hills MULTI Software

You have established the connection to a processor and board. Using three methods you learned about the hardware, and whether it was running. Next, give the processor something to do.

In this part of the tutorial, you load the executable code for the CPU in the IDE. Embedded IDE Link software includes a tutorial project file for Green Hills MULTI. Through the next commands in the tutorial, you locate the

tutorial project file and load it into Green Hills MULTI. The `open` method directs Green Hills MULTI to load a project file or workspace file.

---

**Note** To continue the tutorial, you must identify or create a directory to which you have write access. Embedded IDE Link software cannot create a directory for you. Create one in the Microsoft Windows directory structure before you proceed with the this tutorial.

---

Green Hills MULTI has its own workspace and workspace files that are quite different from MATLAB workspace files and the MATLAB workspace. Remember to monitor both workspaces. To change the working directory to your writable directory:

- 1 Use `cd` to switch to the writable directory

```
prj_dir=cd('C:\ide_link_mu_demo')
```

where the name and path to the writable directory is a string, such as `C:\ide_link_mu_demo` as used in the example. Replace `C:\ide_link_mu_demo` with the full path to your writable directory.

- 2 Change your working directory to the new directory by entering the following command:

```
cd(id,prj_dir)
```

- 3 Use the following command to create a new Green Hills MULTI project named `debug_demo.gpj` in the new directory:

```
new(id, 'debug_demo.gpj')
```

Switch to the IDE to verify that your new project exists. Next, add source files to your project.

- 4 Add the provided source file—`multilinkautointttutorial.c` to the project `debug_demo.gpj` using the following command:

```
add(id, 'multilinkautointttutorial')
```

- 5 Save your project.

```
save(id,'my_debug_demo.gpj','project')
```

Your IDE project is saved with the name `my_debug_demo.gpj` in your writable directory. The input string 'project' specifies that you are saving a project file.

- 6 Next, set the build options for your project. Use the following command to set the compiler build options to use and specify a processor (optional).

```
setbuilddopt(id,'Compiler','-G -cpu=V850')
```

The input argument `-cpu=V850` is optional to specify the processor. Change to processor designation to match your processor if necessary.

## Running the Project

After you create `dot_project_c.gpj` in the IDE, you can use Embedded IDE Link software functions to create executable code from the project and load the code to the processor.

To build the executable and download and run it on your processor:

- 1 Use the following build command to build an executable module from the project `debug_demo.gpj`.

```
build(id,'all',20) % Set optional time-out period to 20 seconds.
```

- 2 To load the new executable to the processor, use `load` with the project file name and the object name. The name of the executable is `debug_demo`.

```
load(id,'debug_demo',30); % Set time-out value to 30 seconds.
```

Embedded IDE Link software provides methods to control processor execution—`run`, `halt`, and `reset`. To demonstrate these methods, use `run` to start the program you just loaded on to the processor, and then use `halt` to stop the processor.

- 1 Enter the following methods at the command prompt and review the response in the MATLAB command window.

```
run(id)      % Start the program running on the processor.
halt(id)     % Halt the processor.
reset(id)    % Reset the program counter to start of program.
```

Use `isrunning` after the `run` method to verify that the processor is running. After you stop the processor, `isrunning` can verify that the processor has stopped.

## Working With Data in Memory

Embedded IDE Link software provides methods that enable you to read and write data to memory on the processor. Reading and writing data depends on the symbol table for your project. The symbol table is available only after you load the executable into the debugger. This section introduces `address` and `dec2hex`. Use them to read the addresses of two global variables—`ddat` and `idat`.

- 1 After you load `debug_demo` into the debugger, enter the following commands to read the addresses of `ddat` and `idat`:

```
ddatA=address(id, 'ddat')
ddatA =
    3145744         0

ddatI=address(id, 'idat')

ddatI =

    3145728         0
```

- 2 Review the results in hexadecimal representation.

```
dec2hex(ddatA)

ans =

    300010
    000000

dec2hex(ddatI)
```

```
ans =  
  
300000  
000000
```

After you load the target code to the processor, you can examine and modify data values in memory, as the previous `read` function examples demonstrated.

For non-changing data values in memory (static values), the values are available immediately after you load the program file.

A more interesting case is looking at variable values that change during program execution. Manipulating changing data values at intermediate points during execution can provide helpful analysis and verification information.

To enable you to read and write data while your program is running, the software provides methods to insert and delete breakpoints in the source programs. Inserting breakpoints lets you pause program execution to read or change variable data values. You cannot change values while your program is running.

The method `insert` creates a new breakpoint at either a source file locations, such as a line number, or at a physical memory address. `insert` takes either the line number or the address as an input argument.

To read the values in the next section of this tutorial, use the following methods to insert breakpoints at lines 24 and 29 in the source file `multilinkautointtutorial.c`

- 1 Change directories to your original working directory.

```
cd(id,proj_dir);
```

- 2 (Optional for convenience) Create variables for the line numbers in the source file.

```
brkpt24 = 24;  
brtpt29 = 29;
```

- 3 Use the following commands to insert breakpoints on line 24 and line 29 of the source file:

```
insert(id,'multilinkautointttutorial',brkpt24); % Insert breakpoint on line 24.  
insert(id,'multilinkautointttutorial',brkpt29); % Insert breakpoint on line 29.
```

- 4** Open and activate the file in the IDE from the MATLAB command window by issuing the following commands:

```
open(id,'multilinkautointttutorial');  
activate(id,'multilinkautointttutorial');
```

Activating `multilinkautointttutorial.c` transfers focus in the IDE to the activated file. Switch to the IDE to verify that the file is in your project and open.

When you look in the IDE debugger window, the breakpoints you added to `multilinkautointttutorial.c` are marked by a STOP sign icon on lines 24 and 29.

A similar method, `remove`, deletes breakpoints.

To help you inspect the source file in the IDE and verify the breakpoints, the `open` and `activate` methods display the file `multilinkautointttutorial.c` in the IDE and force the source file to the front.

One final method actually connects the IDE to your hardware or simulator. `connect` takes a `ghsmulti` object as an input argument to connect the specific IDE primary target referenced by `id` to the associated processor.

## More Memory Data Manipulation

The source file `multilinkaaautointttutorial.c` defines two 1-by-4 global data arrays—`ddat` and `idat`. You can locate the declaration in the file. Embedded IDE Link software provides the read and write methods so you can access the arrays from MATLAB. Find the declaration and note the initialization values.

This tutorial section demonstrates reading and writing data in memory, and controlling the processor.

- 1** Get the address of the symbols `ddat` and `idat`. Enter the following commands at the prompt.



```
ddat_addr=address(id,'ddat'); % Get address from symbol table.
idat_addr=address(id,'idat');
```

- 2** Create two MATLAB variables to specify the data types for ddat and idat.

```
ddat_type='double';
idat_type='int32';
```

- 3** Declare some values in two MATLAB variables.

```
ddat_value=double([pi 12.3 exp(-1) sin(pi/4)]);
idat_value=int32(1:4);
```

- 4** Stop the processor.

```
halt(id)
```

- 5** Reload the project. If you did not save the source file in the project, reloading the project removes the breakpoints you added and move the program counter (PC) to the start of the program.

```
% Reload program file (.gpj). Reset PC to program start.
reload(id,100);
```

- 6** Use the following commands to restore the breakpoints on line 24 and 29.

```
insert(id,'multilinkautointtutorial.c',brkpt24);
insert(id,'multilinkautointtutorial.c',brkpt29);
```

- 7** Use the following method to connect the IDE to the processor:

```
connect(id);
```

- 8** With the breakpoints in the code, run the program until it stops at the first breakpoint on line 24.

```
run(id,'runtohalt',30); % Set time-out to 30 seconds.
```

- 9** Check the current values stored in ddat and idat. Later in this tutorial you change these values from MATLAB.

```
% Do ddat values match initialization values in the source?
ddatV=read(id,address(id,'ddat',ddat_type,4))
```

```
idatV=read(id,address(id,'idat'),idat_type,4)
```

MMATLAB displays the values of ddatV and idatV.

```
ddatV=
```

```
16.300   -2.1300   5.1000   11.8000
```

```
idatV=
```

```
1 508   646   7000
```

- 10** Change the values in ddat and idat by writing new values to the memory addresses.

```
% Write pi, 12.3, exp(-1), and .7070 to memory.
write(id,address(id,'ddata'),ddat_value)
% Write vector [1:4] to memory.
write(id,address(id,'idat'),idat_value)
```

- 11** Resume the program execution from the breakpoint and run until the program stops.

```
run(id,'runtohalt','30'); % Stop at next breakpoint (line 29).
```

- 12** Read the values in memory for ddat and idat to verify the changes.

```
% Read the data as double data type.
ddatV = read(id,address(id,'ddat'),ddat_type,4)
```

```
ddatV=
```

```
3.1416   12.3000   0.3679   0.7071
```

```
% Read the data as int32 data type.
```

```
idatV = read(id,address(id,'idat'),idat_type,4)
```

```
idatV=
```

```
1 2 3 4
```

The data stored in ddat and idat are what you wrote to memory.

- 13** After you review the data, restart the processor to run to return the PC to the program start.

```
restart(id);
```

## Closing the Connections to Green Hills MULTI Software

Objects that you create in Embedded IDE Link software have connections to Green Hills MULTI IDE. Until you delete these objects, the Green Hills MULTI process (`Idde.exe` in the Windows Task Manager) remains in memory. Closing MATLAB removes these objects automatically, but there may be times when it helps to delete the handles manually, without quitting MATLAB.

---

**Note** When you clear the last `ghsmulti` object, the software does not close the running Embedded IDE Link service. When it does close the IDE, it does not save current projects or files in the IDE, and it does not prompt you to save them.

---

A best practice is to save your projects and files before you clear `ghsmulti` objects from your MATLAB workspace.

Use the following commands to close the project files in Green Hills MULTI IDE and remove the breakpoints you added to the source file.

```
close(id,'debug_demo.gpj','project') % Close the project file.
remove(id,'multilinkautointtutorial.c',brkpt24);

remove(id,'multilinkautointtutorial.c',brkpt29);
```

Finally, to delete your link to Green Hills MULTI use `clear id`.

You have completed the Automation Interface tutorial using Embedded IDE Link software.

## Tasks Performed During the Tutorial

During the tutorial you performed the following tasks:

- 1** Created and queried objects that refer to a session in Embedded IDE Link software to get information about the session and processor.
- 2** Used MATLAB software to load files into the Green Hills MULTI IDE and used methods in MATLAB software to run that file.
- 3** Closed the links you opened to Green Hills MULTI software.

This set of tasks is used in any development work you do with signal processing applications. Thus, the tutorial gives you a working process for using Embedded IDE Link software and your signal processing programs to develop programs for a range of processors.

## Constructing Objects

When you create a connection to a session in Green Hills MULTI using the `ghsmulti` function, you create a `ghsmulti` object (in object-oriented design terms, you *instantiate* the `ghsmulti` object). The object implementation relies on MATLAB object-oriented programming capabilities like the objects in MATLAB or Filter Design Toolbox™ software.

The discussions in this section apply to the objects in Embedded IDE Link software. Because `ghsmulti` objects use the MATLAB software techniques, the information about working with the objects, such as how you get or set object properties or use methods, apply to the `ghsmulti` objects in Embedded IDE Link software.

Like other MATLAB structures, `ghsmulti` objects have predefined fields referred to as *object properties*.

You specify object property values by the following methods:

- Specifying the property values when you create the object
- Creating an object with default property values, and changing some or all of these property values later

For examples of setting link properties, refer to “Setting Property Values with `set`”.

### Example – Constructor for `ghsmulti` Objects

The easiest way to create an object is to use the function `ghsmulti` to create an object with the default properties. Create an object named `id` referring to a session in Green Hills MULTI by entering the following syntax:

```
id = ghsmulti
```

MATLAB responds with a list of the properties of the object `id` you created along with the associated default property values.

```
MULTI Object:
  Host Name      : localhost
  Port Num      : 4444
```

```
Default timeout : 10.00 secs  
MULTI Dir      : C:\ghs\multi500\ppc\
```

The object properties are described in the `ghsmulti` documentation.

---

**Note** These properties are set to default values when you construct links.

---

# Properties and Property Values

## In this section...

- “Working with Properties” on page 2-21
- “Setting and Retrieving Property Values” on page 2-21
- “Setting Property Values Directly at Construction” on page 2-22
- “Setting Property Values with set” on page 2-22
- “Retrieving Properties with get” on page 2-23
- “Direct Property Referencing to Set and Get Values” on page 2-23
- “Overloaded Functions for ghsmulti Objects” on page 2-24

## Working with Properties

Links (or objects) in this Embedded IDE Link software have properties associated with them. Each property is assigned a value. You can set the values of most properties, either when you create the link or by changing the property value later. However, some properties have read-only values. Also, a few property values, such as the board number and the processor to which the link attaches, become read-only after you create the object. You cannot change those after you create your link.

## Setting and Retrieving Property Values

You can set ghsmulti object property values by either of the following methods:

- Directly when you create the link — see “Setting Property Values Directly at Construction”
- By using the set function with an existing link — see “Setting Property Values with set”

Retrieve ghsmulti object property values with the get function.

Direct property referencing lets you either set or retrieve property values for ghsmulti objects.

## Setting Property Values Directly at Construction

To set property values directly when you construct an object, include the following entries in the input argument list for the constructor method `ghsmulti`:

- A string for the property name to set, followed by a comma. Enclose the string in single quotation marks as you do any string in MATLAB.
- The property value to associate with the named property. Sometimes this value is also a string.

You can include as many property names in the argument list for the object construction command as there are properties to set directly.

### Example — Setting Link Property Values at Construction

Create a connection to an instance of the IDE in Green Hills MULTI software and set the following object properties:

- Link to the specified MULTI instance and host.
- Specify the communication port on the host.
- Set the global timeout to 5 s. The default is 10 s.

Set these properties when you construct the object by entering

```
id = ghsmulti('hostname','localhost','portnum',4444,'timeout',5);
```

The `localhost`, `portnum`, and `timeout` properties are described in Link Properties, as are the other properties for links.

## Setting Property Values with `set`

After you construct an object, the `set` function lets you modify its property values.

Using the `set` function, you can Set link property values.



### Example — Setting Link Property Values Using set

To set the timeout specification for the link `id` from the previous section, enter the following syntax:

```
set(id,'timeout',8);

get(id,'timeout');
ans=

     8
```

The display reflects the changes in the property values.

### Retrieving Properties with get

You can use the `get` command to retrieve the value of an object property.

### Example — Retrieving Link Property Values Using get

To retrieve the value of the `hostname` property for `id`, and assign it to a variable, enter the following syntax:

```
host=get(id,'hostname')

host =

localhost
```

### Direct Property Referencing to Set and Get Values

You can directly set or get property values using MATLAB structure-like referencing. Do this by using a period to access an object property by name, as shown in the following example.

### Example — Direct Property Referencing in Links

To reference an object property value directly, perform the following steps:

- 1 Create a link with default values.
- 2 Change its time out and number of open channels.

```
id = ghsmulti;  
id.time = 6;
```

## **Overloaded Functions for ghsmulti Objects**

Several methods and functions in Embedded IDE Link software have the same name as functions in other MathWorks products. These functions behave similarly to their original counterparts, but you apply them to an object. This concept of having functions with the same name operate on different types of objects (or on data) is called *overloading* of functions.

For example, the `set` command is overloaded for objects. After you specify your object by assigning values to its properties, you can apply the methods in this toolbox (such as `address` for reading an address in memory) directly to the variable name you assign to your object. You do not have to specify your object parameters again.

For a list of the methods that act on `ghsmulti` objects, refer to the “Functions — Alphabetical List” in the function reference pages.

## ghsmulti Object Properties

### In this section...

“Quick Reference to ghsmulti Properties” on page 2-25

“Details About ghsmulti Object Properties” on page 2-25

### Quick Reference to ghsmulti Properties

The following table lists the properties for the links in Embedded IDE Link software. The second column indicates to which object the property belongs. Knowing which property belongs to each object in an interface tells you how to access the property.

Property Name	User Settable?	Description
hostname	At construction only	Reports the name of the host the Embedded IDE Link service in Green Hills MULTI that the object references.
portnum	At construction only	Stores the number of the port to communicate with MULTI.
timeout	Yes/default	Contains the global timeout setting for the link.

Some properties are read only. Thus, you cannot set the property value. Other properties you can change at any time. If the entry in the User Settable column is “At construction only,” you can set the property value only when you create the object. Thereafter, it is read only.

### Details About ghsmulti Object Properties

To use the objects for Green Hills MULTI interface, set values for the following:

- `hostname` — Specify the session with which the object interacts.
- `portnum` — Specify the processor in the session. If the board has multiple processors, `procnum` identifies the processor to use.

- `timeout` — Specify the global timeout value. (Optional. Default is 10 s.)

Details of the properties associated with `ghsmulti` objects appear in the following sections, listed in alphabetical order by property name.

### **hostname**

Property `hostname` identifies the host that is running the Embedded IDE Link service. Use `hostname` to specify the machine to host your service.

To work with a service, you need the `hostname` and `portnum` values. `Hostname` supports the string `localhost` only.

### **portnum**

Property `portnum` specifies the port for communicating with the Embedded IDE Link service. MATLAB uses sockets to communicate with Green Hills MULTI. The `portnum` property value specifies the port, with a default value of 4444. When you create a new `ghsmulti` object, Embedded IDE Link software assumes the port value is 4444 unless you enter a different value when you configure the software or use the `portnum` input argument with `ghsmulti`.

### **timeout**

Property `timeout` specifies how long Green Hills MULTI waits for any process to finish. You set the global timeout when you create an object for a session in Green Hills MULTI. The default global timeout value 10 s. The following example shows the `timeout` value for object `id2`.

```
display(id2)

MULTI Object:
  Host Name      : localhost
  Port Num      : 4444
  Default timeout : 10.00 secs
  MULTI Dir     : C:\ghs\multi500\ppc\
```

# Project Generator

---

- “Introducing Project Generator” on page 3-2
- “Project Generator Tutorial” on page 3-3
- “Model Reference” on page 3-11

## Introducing Project Generator

Project generator provides the following features for developing projects and generating code:

- Automated project building for Green Hills MULTI that lets you create MULTI projects from code generated by Real-Time Workshop and Real-Time Workshop Embedded Coder. Project generator populates projects in the MULTI development environment.
- Blocks in the library `idelinklib_ghsmulti` for controlling the scheduling and timing in generated code.
- Highly configurable code generation using model configuration parameters and target preferences block options.
- Ability to use Embedded IDE Link software with one of two system target files to generate code specific to your processor.
- Highly configurable project build process.
- Automatic downloading and running of your generated projects on your processor.

To configure your Simulink models to use the Project Generator component, do one or both of the following tasks:

- Add a Target Preferences block from the Embedded IDE Link blockset (`idelinklib_ghsmulti`) to the model.
- To use the asynchronous scheduler capability in Embedded IDE Link software, add a hardware interrupt block or idle task block.

The following sections describe the blockset and the blocks in it, the scheduler, and the Project Generator component.

# Project Generator Tutorial

## In this section...

“Process for Building and Generating a Project” on page 3-3

“Create the Model” on page 3-4

“Adding the Target Preferences Block to Your Model” on page 3-5

“Specifying Simulink Configuration Parameters for Your Model” on page 3-7

“Creating Your Project” on page 3-9

## Process for Building and Generating a Project

In this tutorial, you build a model and generate a project from the model into Green Hills MULTI.

---

**Note** The model shows project generation only. You cannot build and run the model on your processor without additional blocks.

---

To generate a project from a model, complete the following tasks:

- 1** Use Simulink blocks, Signal Processing Blockset blocks, and blocks from other blocksets to create the model application.
- 2** Add the target preferences block from the Embedded IDE Link Target Preferences library to your model.
- 3** Double-click the Target Preferences block to open the block dialog box.
- 4** Select your processor from the **Processor** list. Verify and set the block parameters for your hardware, such as **CPU clock** and the options on the **Memory** and **Section** panes. In most cases, the default settings for the selected processor work fine.
- 5** Set the configuration parameters for your model, including the following parameters:

- Solver parameters such as simulation start and solver options. Choose the discrete solver when you generate executables. If you are using PIL, select any setting from the **Type** and **Solver** lists.
- Real-Time Workshop options such as processor configuration and processor compiler selection

6 Generate your project.

7 Review your project in MULTI.

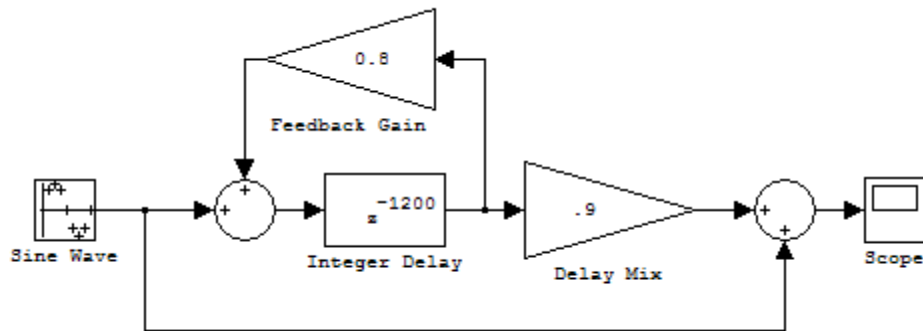
## Create the Model

To build the model for this tutorial, follow these steps:

1 Start Simulink.

2 Create a new model by selecting **File > New > Model** from the **Simulink** menu bar.

3 Use Simulink blocks and Signal Processing Blockset blocks to create the following model.



Look for the Integer Delay block in the Discrete library of Simulink and the Gain block in the Commonly Used Blocks library. This model implements an audio signal reverberation scheme. Part of the input audio signal passes directly to the output. A delayed version passes through a feedback loop



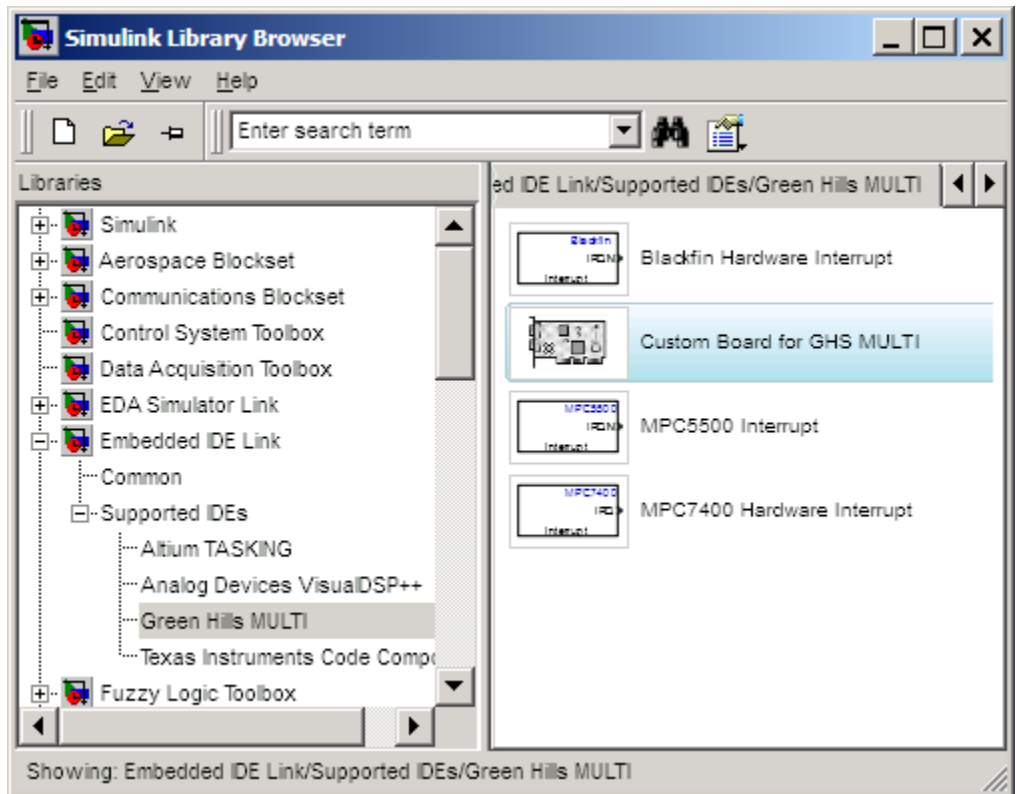
before reaching the output. The result is an echo, or reverberation, added to the audio output.

4 Save your model with a suitable name before continuing.

## Adding the Target Preferences Block to Your Model

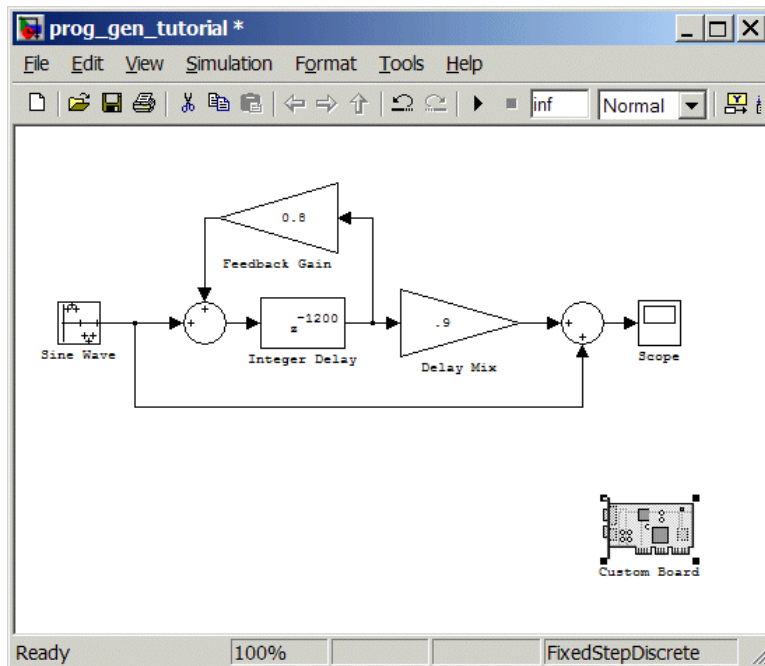
To configure your model to work with the processors your IDE supports, add a target preferences block to your model.

Use the Target Preferences/Custom Board for GHS MULTI block, located in the `idelinklib_ghsmulti` block library.



To configure the Target Preferences/Custom Board for GHS MULTI (the “Custom Board”) block in your model:

- 1 Double-click Embedded IDE Link in the Simulink Library browser to open the `idelinklib_ghsmulti` blockset.
- 2 Double-click the library Target Preferences to see the Custom Board block.
- 3 Drag and drop the Custom Board block to your model as shown in the following figure.



- 4 Double-click the Custom Board block to open the block dialog box.
- 5 In the Block dialog box, select your processor from the **Processor** list.
- 6 Check the **CPU clock** value and change it if necessary to match your processor clock rate.
- 7 Review the settings on the **Memory** and **Sections** tabs to verify that they are correct for the processor you selected.
- 8 Click **OK** to close the Target Preferences dialog box.

You have completed the model. Next, configure the model configuration parameters to generate a project in Green Hills MULTI from your model.

## Specifying Simulink Configuration Parameters for Your Model

The following sections describe how to configure the build and run parameters for your model. Generating a project, or building and running a model on the processor, starts with configuring model options in the Configuration Parameters dialog box in Simulink.

### Setting Solver Options

After you have designed and implemented your digital signal processing model in Simulink, complete the following steps to set the configuration parameters for the model:

- 1 Open the Configuration Parameters dialog box and set the appropriate options on the **Solver** category for your model and for Embedded IDE Link software.
  - Set **Start time** to 0.0 and **Stop time** to `inf` (model runs without stopping). If you set a stop time, your generated code does not honor the setting. Set this parameter to `inf` for completeness.
  - Under **Solver options**, select the **fixed-step** and **discrete** settings from the lists. When you use PIL, select any setting on the **Type** and **Solver** lists.
  - Set the **Fixed step size** to **Auto** and the **Tasking Mode** to **Single Tasking**.

---

**Note** Generated code does not honor Simulink stop time from the simulation. Stop time is interpreted as `inf`. To implement a stop in generated code, you must put a Stop Simulation block in your model.

---

Ignore the **Data Import/Export**, **Diagnostics**, and **Optimization** categories in the Configuration Parameters dialog box. The default settings are correct for your new model.

## Setting Real-Time Workshop Code Generation Options

To configure Real-Time Workshop software to use the correct processor files, compile your model, and run your model executable file, set the options in the Real-Time Workshop category of the model Configuration Parameters. Follow these steps to set the Real-Time Workshop software options to generate code tailored for your processor:

- 1 Select Real-Time Workshop on the **Select** tree.
- 2 In **Target selection**, click **Browse** to select the appropriate system target file for code generation—`multilink_grt.tlc` or `multilink_ert.tlc` (if you use Real Time Workshop Embedded Coder software). The correct target file might already be selected.

Clicking **Browse** opens the **System Target File Browser** to allow you to change the system target file.

- 3 On the **System Target File Browser**, select the proper system target file `multilink_grt.tlc` or `multilink_ert.tlc`, and click **OK** to close the browser.

## Setting Embedded IDE Link Code Generation Options

After you set the Real-Time Workshop options for code generation, set the options that apply to your Embedded IDE Link software run-time and build processes.

- 1 From the **Select** tree, choose Embedded IDE Link to specify code generation options that apply to the processor.
- 2 Set the following **Runtime** options:
  - **Build action:** `Create_project`.
  - **Interrupt overrun notification method:** `Print_message`.
- 3 (optional) Under **Link Automation**, verify that **Export MULTI link handle to base workspace** is selected and provide a name for the handle in **MULTI link handle name**.
- 4 If you are using an actual board, identify a Board Support Package (BSP) in the **Compiler options string** (under **Project options**). For example,

enter “-bsp=at91rm9200”. If you do not provide this type of information, the software can generate errors that do not identify the absence of linker directives as the cause.

- 5** Under **Code Generation**, clear all of the options.
- 6** Change the category on the **Select** tree to **Hardware Implementation**.
- 7** Verify that the **Device** type is the correct value for your processor—**Analog Devices, NEC, or Freescale**.

You have configured the Real-Time Workshop options that let you generate a project for your processor. A few Real-Time Workshop categories on the **Select** tree, such as **Comments, Symbols, and Optimization** do not require configuration for use with Embedded IDE Link software. In some cases, set options in the other categories to configure other code generation features.

For your new model, the default values for the options in these categories are correct. For other models you develop, setting the options in these categories provides more information during the build process. Some of the options configure the model to run TLC debugging when you generate code. Refer to your Simulink and Real-Time Workshop documentation for more information about setting the configuration parameters.


## Creating Your Project


After you set the configuration parameters and configure Real-Time Workshop to create the files you need, you direct Real-Time Workshop to create your project:

- 1** Click **OK** to close the Configuration Parameters dialog box.
- 2** To verify that you configured your Embedded IDE Link software correctly, issue the following command at the prompt to open the Embedded IDE Link Configuration dialog box.

```
ghsmulticonfig
```

- 3** Verify the settings in the Embedded IDE Link dialog box.
- 4** After you verify the settings, click **OK** to close the dialog box.

- 5 Enter `cd` at the prompt to verify that your working directory is the right one to store your project results.
- 6 Click **Incremental Build** () on the model toolbar to generate your project into Green Hills MULTI IDE.

When you press  with `Create_project` selected for **Build action**, the build process starts the Green Hills MULTI application and populates a new project.

# Model Reference

## In this section...

“About Model Reference” on page 3-11

“How Model Reference Works” on page 3-11

“Using Model Reference” on page 3-12

“Configuring Targets to Use Model Reference” on page 3-14

## About Model Reference

Model reference lets your model include other models as modular components. This technique is useful because it provides the following capabilities:

- Simplifies working with large models by letting you build large models from smaller ones, or even large ones.
- Lets you generate code once for all the modules in the entire model and then only regenerate code for modules that change.
- Lets you develop the modules independently.
- Lets you reuse modules and models by reference, rather than including the model or module multiple times in your model. Also, multiple models can refer to the same model or module.

Your Real-Time Workshop documentation provides much more information about model reference.

## How Model Reference Works

Model reference behaves differently in simulation and in code generation. This discussion uses the following terms:

- The *Top model* is the root model block or model. It refers to other blocks or models. In the model hierarchy, this model is the topmost model.
- *Referenced models* are blocks or models that other models reference, such as models the top model refers to. All models or blocks below the top model in the hierarchy are reference models.

The following sections describe briefly how model reference works. More details are available in your Real-Time Workshop documentation in the online Help system.

### **Model Reference in Simulation**

When you simulate the top model, Real-Time Workshop detects that your model contains referenced models. Simulink generates code for the referenced models and uses the generated code to build shared library files for updating the model diagram and simulation. It also creates an executable (.mex file) for each reference model that is used to simulate the top model.

When you rebuild reference models for simulations or when you run or update a simulation, Simulink rebuilds the model reference files. Whether reference files or models are rebuilt depends on whether and how you change the models and on the **Rebuild options** settings. You can access these settings through the **Model Reference** pane of the Configuration Parameters dialog box.

### **Model Reference in Code Generation**

Real-Time Workshop requires executables to generate code from models. If you have not simulated your model at least once, Real-Time Workshop creates a .mex file for simulation.

Next, for each referenced model, the code generation process calls `make_rtw` and builds each referenced model. This build process creates a library file for each of the referenced models in your model.

After building all the referenced models, Real-Time Workshop calls `make_rtw` on the top model. The call to `make_rtw` links to the library files Real-Time Workshop created for the associated referenced models.

### **Using Model Reference**

With few limitations or restrictions, Embedded IDE Link software provides full support for generating code from models that use model reference.

### **Build Action Setting**

The most important requirement for using model reference with the Green Hills MULTI software supported processors is you must set the **Build action**



(select **Configuration Parameters > Embedded IDE Link**) for all models referred to in the simulation to `Archive_library`.

To set the build action, perform the following steps:

- 1 Open your model.
- 2 Select **Simulation > Configuration Parameters** from the model menus.  
The Configuration Parameters dialog box opens.
- 3 From the **Select** tree, choose Embedded IDE Link.
- 4 In the right pane, under **Runtime**, select set `Archive_library` from the **Build action** list.

If your top model uses a reference model that does not have the build action set to `Archive_library`, the build process automatically changes the build action to `Archive_library` and issues a warning about the change.

Selecting `Archive_library` disables the **Interrupt overrun notification method**, **Export MULTI link handle to the base workspace**, and **System stack size** options for the referenced models.

### Target Preferences Blocks in Reference Models

Each referenced model and the top model must include a Target Preferences block for the correct processor. Configure all the Target Preferences blocks for the same processor.

The referenced models need target preferences blocks to provide information about which compiler and which archiver to use. Without these blocks, the compile and archive processes do not work.

By design, model reference does not allow information to pass from the top model to the referenced models. Referenced models must contain all the necessary information, which the Target Preferences block in the model provides.

### **Other Block Limitations**

Model reference with Embedded IDE Link software code generation options does not allow you to use noninlined S-functions in reference models. Verify that the blocks in your model do not use noninlined S-functions.

### **Configuring Targets to Use Model Reference**

When you create models to use in Model Referencing, keep in mind the following considerations:

- Your model must use a system target file derived from the ERT or GRT target files.
- When you generate code from a model that references other models, configure the top-level model and the referenced models for the same system target file.
- Real-Time Workshop builds and Embedded IDE Link software projects do not support external mode in model reference. If you select the external mode option, it is ignored during code generation.
- Your TMF must support use of the shared utilities directory, as described in Supporting Shared Utility Directories in the Build Process in the Real-Time Workshop documentation.

To use an existing processor, or a new processor, with Model Reference, set the `ModelReferenceCompliant` flag for the processor. For information about setting this option, refer to `ModelReferenceCompliant` in the online Help system.

If you start with a model that was created before MATLAB release R14SP3, use the following command to make your model compatible with model reference :

```
% Set the Model Reference Compliant flag to on.  
set_param(bdroot, 'ModelReferenceCompliant', 'on')
```

Code that you generate from Simulink models by using Embedded IDE Link software includes the model reference capability. You do not need to set the flag.

# Block Reference

---

Block Library: idelinklib\_ghsmulti    Blocks for Green Hills MULTI  
(p. 4-2)

## **Block Library: idelinklib\_ghsmulti**

Blackfin Hardware Interrupt

MPC5500 Interrupt

MPC7400 Hardware Interrupt

Generate Interrupt Service Routine

Generate Interrupt Service Routine

Generate Interrupt Service Routine

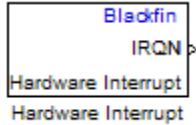
# Blocks — Alphabetical List

---

# Blackfin Hardware Interrupt

**Purpose** Generate Interrupt Service Routine

**Library** Block Library: idelinklib\_ghsmulti



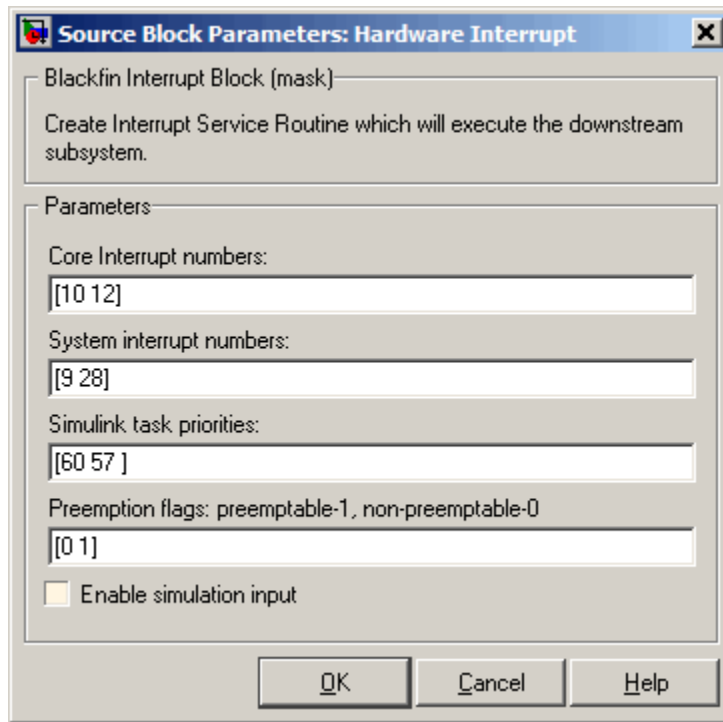
**Description** Create interrupt service routines (ISR) in the software generated by the build process. When you incorporate this block in your model, code generation results in ISRs on the processor that run the processes that are downstream from this block or an Idle Task block connected to this block. Core interrupts trigger the ISRs. System interrupts trigger the core interrupts. In the following figure, you see the mapping possibilities between system interrupts and core interrupts.

## Interrupts

Blackfin processors support the interrupt numbers shown in the following table. Some Blackfin processors do not support all of the system interrupts.

Interrupt Description	Valid Range in Parameter
Core interrupt numbers	7 to 15
System interrupt numbers	0 to 31 (The upper end value depends on the processor. May be less than 31.)

## Dialog Box



### Core interrupt numbers

Specify a vector of one or more interrupt numbers for the interrupt service routines (ISR) to install. The valid range is 7 to 15, where 7 through 13 are hardware driven, and 14 and 15 are software driven. Core interrupts numbered 0 to 6 are reserved and cannot be entered in this field.

The width of the block output signal corresponds to the number of interrupt values you specify in this field. Triggering of each ISR depends on the core interrupt value, the system interrupt value, and the preemption flag you enter for each interrupt. These three values define how the code and processor respond to interrupts during asynchronous scheduler operations.

# Blackfin Hardware Interrupt

---

## System interrupt numbers

System interrupt numbers identify system interrupts to map to core interrupts. Enter one or more values as a vector. The valid range is 0 through 31, although the valid range depends on your processor. Some processors do not support the full range of 32 system interrupts. The software does not test for valid system interrupt values. You must verify that your values are valid for your processor. You must specify at least one system interrupt number to use asynchronous scheduling.

The block maps the first interrupt value in this field to the first core interrupt value you enter in **Core interrupt numbers**, it maps the second system interrupt value to the second core interrupt value, and so on until it has mapped all of the system interrupt values to core interrupt values. You cannot map more than one system interrupt to the same core interrupt. Therefore, you can enter one system interrupt value in this field and map it to more than one core interrupt. You cannot enter more than one value in this field and map the values to one core interrupt.

When you trigger one of the system interrupts in this field, the block triggers the ISR associated with the core interrupt that is mapped to the system interrupt.

## Simulink task priorities

Each output of the Hardware Interrupt block drives a downstream block (for example, a function call subsystem). Simulink task priority specifies the Simulink priority of the downstream blocks. Specify an array of priorities corresponding to the interrupt numbers entered in **Interrupt numbers**.

Proper code generation requires rate transition code (see Rate Transitions and Asynchronous Blocks). The task priority values ensure absolute time integrity when the asynchronous task must obtain real time from its base rate or its caller. Typically, assign priorities for these asynchronous tasks that are higher than the priorities assigned to periodic tasks.



## **Preemption flags: preemptable – 1, non-preemptable – 0**

Higher priority interrupts can preempt interrupts that have lower priority. To control this preemption, use the preemption flags to specify whether an interrupt can be preempted.

- Entering 1 indicates the corresponding core interrupt can be preempted.
- Entering 0 indicates the corresponding interrupt cannot be preempted.

When **Core interrupt numbers** contains more than one interrupt priority, you can assign different preemption flags to each interrupt by entering a vector of preemption flag values that correspond to the order of the interrupts in **Core interrupt numbers**. If **Core interrupt numbers** contains more than one interrupt, and you enter only one flag value in this field, that status applies to all interrupts.

For example, the default settings [0 1] indicate that the interrupt with value 10 in **Core interrupt numbers** is not preemptible and the value 12 interrupt can be preempted.

## **Enable simulation input**

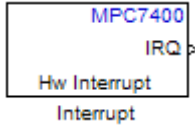
When you select this option, Simulink adds an input port to the Hardware Interrupt block. This port receives input only during simulation. Connect one or more simulated interrupt sources to the simulation input.

# MPC7400 Hardware Interrupt

---

**Purpose** Generate Interrupt Service Routine

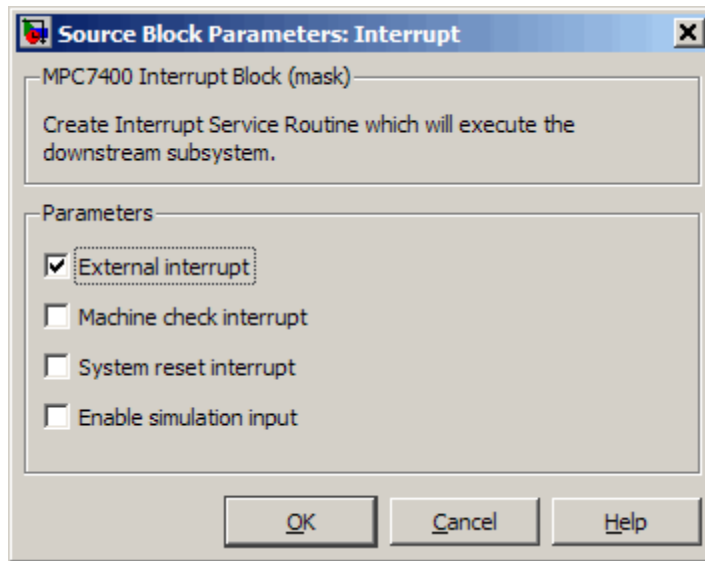
**Library** Block Library: idelinklib\_ghsmulti



**Description** The block creates ISRs for three processor interrupts—External, Machine check and System reset. When you incorporate this block in your model, code generation results in ISRs on the processor that run the blocks downstream from this block. For more information about these interrupts, refer to your MPC7400 documentation.

When you enable more than one interrupt on the block dialog box, the block multiplexes the ISR outputs onto the output port on the block. To resolve the different ISRs, connect the output port IRQ to a Demux block. Connect the demultiplexed outputs to downstream blocks or subsystems. Refer to Examples to see the multiple interrupt configuration in a model.

## Dialog Box



### External interrupt

Interrupt generated by an external system that asserts the intr pin of the 7400 microprocessor.

### Machine check interrupt

Enable the asynchronous, nonmaskable machine check exception provided by the processor. The exception responds to the conditions described in the MPC7400 documentation.

### System reset interrupt

Enable the asynchronous, nonmaskable System interrupt exception provided by the processor. The exception responds to the conditions described in the MPC7400 documentation.

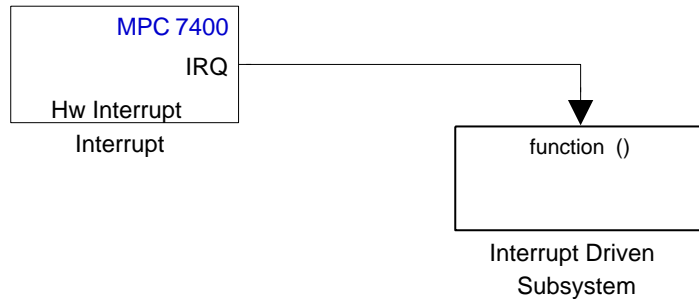
### Enable simulation input

Select this option to have Simulink add an input port to the HW Interrupt block. This port receives input only during simulation. Connect one or more simulated interrupt sources to the input to drive the model interrupt processing.

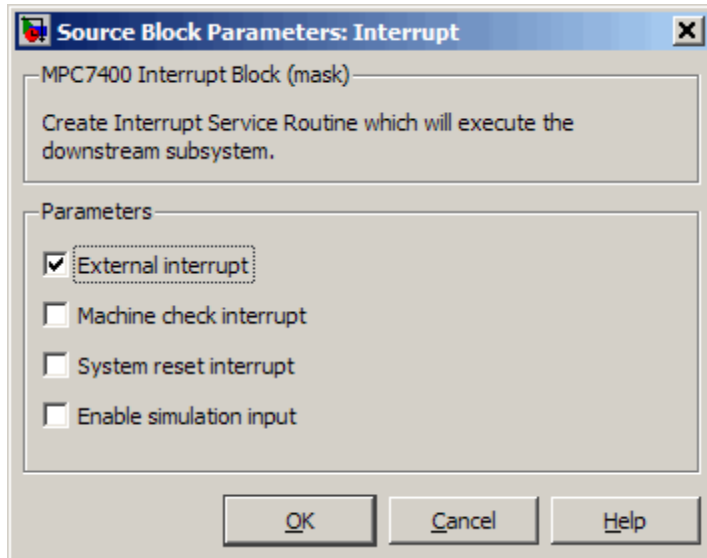
# MPC7400 Hardware Interrupt

## Example

The following model shows the HW Interrupt block triggering a subsystem. The interrupt block is configured to respond to external interrupts.



Here is the block mask.

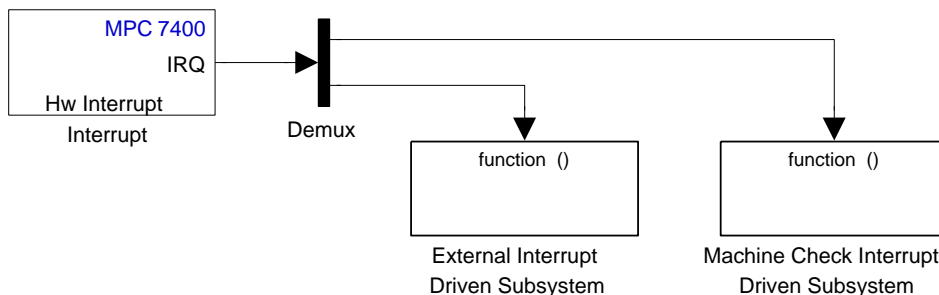


When your peripherals assert the external interrupt pin on the processor, the code generated by the HW Interrupt block during the

# MPC7400 Hardware Interrupt

project build process accepts the interrupt and triggers the attached subsystem through an ISR.

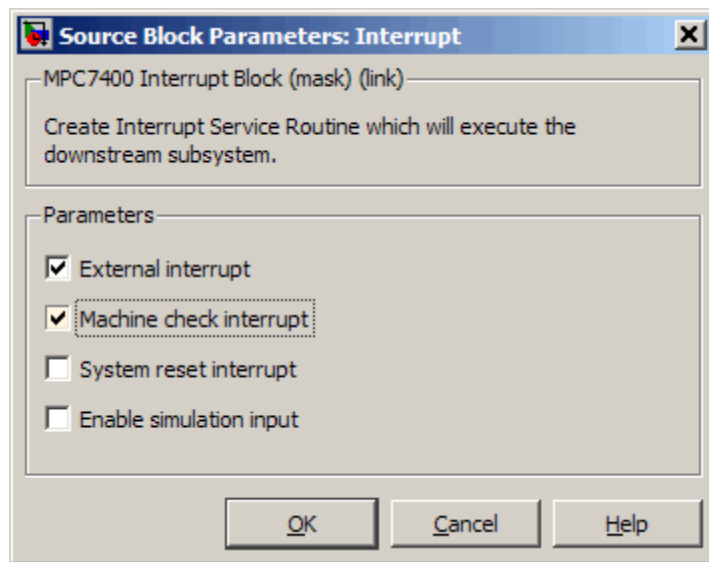
When you select more than one interrupt, connect the output of the block to a Demux block to separate the ISRs, as shown in the following model:



Here is the block mask showing the external and Machine check interrupts selected.

# MPC7400 Hardware Interrupt

---



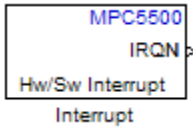
To test your interrupt configuration in simulation, select **Enable simulation input** on the block dialog box and then input a signal to the block to simulate the external interrupt.

## See Also

Idle Task, Memory Allocate, Memory Copy

**Purpose** Generate Interrupt Service Routine

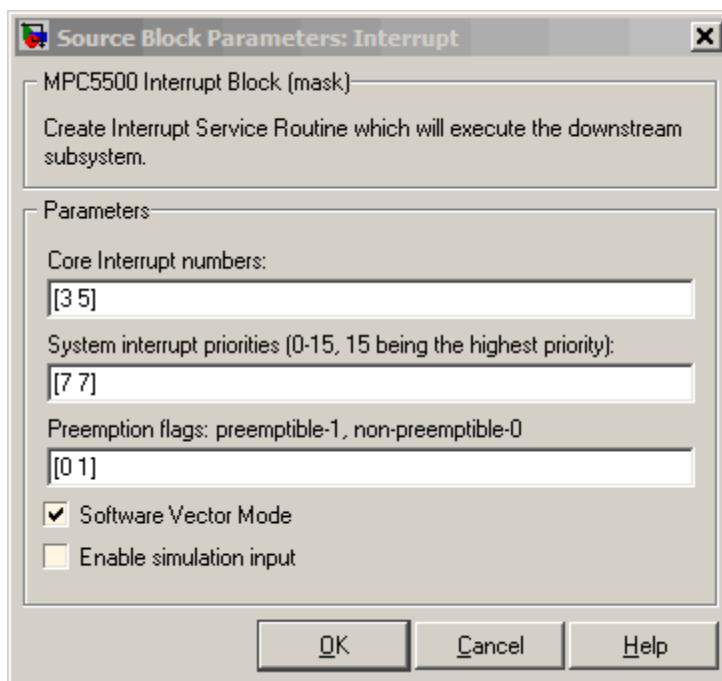
**Library** Block Library: idelinklib\_ghsmulti



**Description** Create interrupt service routines (ISR) in the software generated by the build process. When you incorporate this block in your model, code generation results in ISRs on the processor that either run the processes that are downstream from this block or trigger an Idle Task block connected to this block. Core interrupts trigger the ISRs. System interrupts trigger the core interrupts.

# MPC5500 Interrupt

## Dialog Box



### Core interrupt numbers

Specify a vector of interrupt numbers for the interrupts to install. The block services these interrupts. When your model or code raises one of these interrupts, either through hardware or software, this block reacts to the interrupt and runs the associated downstream block or function. The valid range or interrupts depends on the processor. For example, MPC5553 processors support 212 interrupts. MPC5554 processors support 308 interrupts. Each interrupt in the row vector must be unique. Interrupts that you do not specify in this parameter cause system failures if your project raises them.

The width of the block output signal corresponds to the number of interrupt numbers specified in this field. The values in this



field and the preemption flag entries in **Preemption flags: preemptible-1, non-preemptible-0** define how the code and processor handle interrupts during asynchronous scheduler operations.

## **System interrupt priorities (0–15, 15 being the highest priority)**

Each output of the HW/SW Interrupt block drives a downstream block (for example, a function call subsystem). Simulink task priority specifies the Simulink priority of the downstream blocks. Specify an array of priorities corresponding to the interrupt numbers entered in **Core interrupt numbers**. In the default settings shown in the figure, interrupts 3 and 5 have the same priority value—7.

Proper code generation requires rate transition code (see Rate Transitions and Asynchronous Blocks). The task priority values ensure absolute time integrity when the asynchronous task must obtain real time from its base rate or its caller. Typically, assign priorities for these asynchronous tasks that are higher than the priorities assigned to periodic tasks.

If multiple interrupts share the same priority and are asserted simultaneously, the block selects the lowest numbered interrupt first.

## **Preemption flags: preemptible – 1, non-preemptible – 0**

Higher-priority interrupts can preempt interrupts that have lower priority. To allow you to control preemption, use the preemption flags to specify whether an interrupt can be preempted.

- Entering 1 indicates that the interrupt can be preempted.
- Entering 0 indicates the interrupt cannot be preempted.

You cannot set a task that has priority higher than the base rate to be preemptable.

When **Interrupt numbers** contains more than one interrupt value, you can assign different preemption flags to each interrupt

# MPC5500 Interrupt

---

by entering a vector of flag values to correspond to the order of the interrupts in **Interrupt numbers**. If **Interrupt numbers** contains more than one interrupt, and you enter only one flag value in this field, that status applies to all interrupts.

In the default settings [0 1], the interrupt with priority 5 in **Interrupt numbers** is not preemptible and the priority 8 interrupt can be preempted.

## **Software vector mode**

Select this option to put the block and processor in software vector mode. Enabling this option creates a common interrupt handler. Clearing this option puts the processor in hardware vector mode. Refer to the MULTI documentation for more information about the modes.

## **Enable simulation input**

When you select this option, Simulink adds an input port to the HW/SW Interrupt block. This port is used in simulation only. Connect one or more simulated interrupt sources to the simulation input.

## A

- access properties 2-21
- Archive\_library 3-12

## B

- block limitations using model reference 3-14

## F

- functions
  - overloading 2-24

## G

- getting properties 2-23
- ghsmulti 2-19
- ghsmulti object properties 2-26
  - portnum 2-26
  - procnum 2-26
- Green Hills MULTI® IDE objects
  - tutorial about using 2-2
- Green Hills Software model reference 3-11

## L

- link filters properties
  - getting 2-23
- link properties
  - about 2-25
  - setting 2-23
- link properties, details about 2-25
- links
  - closing Green Hills MULTI® 2-17
  - details 2-25
  - loading files into Green Hills MULTI® IDE 2-9
  - quick reference 2-25
  - working with your processor 2-11

## M

- model reference 3-11
  - about 3-11
  - Archive\_library 3-12
  - block limitations 3-14
  - modelreferencecompliant flag 3-14
  - setting build action 3-12
  - target preferences blocks 3-13
  - using 3-12
- modelreferencecompliant flag 3-14
- MULTI
  - starting from MATLAB 2-4
  - stopping from MATLAB 2-4

## O

- object
  - ghsmulti 2-19
- object properties
  - quick reference table 2-25
- objects
  - creating objects for Green Hills MULTI® IDE 2-8
  - introducing the objects for Green Hills MULTI® IDE tutorial 2-2
  - tutorial about using Automation Interface for Green Hills MULTI® IDE 2-2
- overloading 2-24

## P

- portnum 2-26
- procnum 2-26
- properties
  - link properties 2-25
  - referencing directly 2-23
  - retrieving 2-21
    - function for 2-23
  - retrieving by direct property referencing 2-23
  - setting 2-21

**S**

- set properties 2-21
- start MULTI from MATLAB 2-4
- stop MULTI from MATLAB 2-4
- structure-like referencing 2-23

**T**

- target preferences blocks in referenced models 3-13
- timeout
  - timeout 2-26
- tutorials
  - objects for Green Hills MULTI® 2-2