xPC Target

For Use with Real-Time Workshop®

Modeling

Simulation

Implementation

The MATH WORKS

User's Guide

Version 1

How to Contact The MathWorks:

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup

`support@mathworks.com`	Technical support
`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information

508-647-7000 Phone

508-647-7001 Fax

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

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

xPC Target User's Guide

Printing History:

September 1999	First printing	New for Version 1.0 (Release 11.1)
November 2000	Online only	Revised for Version 1.1 (Release 12.0)
June 2001	Online only	Revised for Version 1.2 (Release 12.1)

Contents

1 Introduction

2 Advanced Procedures

3 Network Communication

4 Embedded Option

5 Environment Reference

6 Target Object Reference

7 Scope Object Reference

1

Introduction

xPC Target has many features. An introduction to these features and the xPC Target software environment will help you develop a model for working with xPC Target.

This chapter includes the following section:

"What Is xPC Target?" - Host-target PC solution for prototyping, testing, and deploying real-time systems

What Is xPC Target?

xPC Target is a host-target solution for prototyping, testing, and deploying real-time systems using standard PC hardware. It is an environment that uses a target PC, separate from the host PC, for running real-time applications.

In this environment you use the desktop computer as a host PC with MATLAB®, Simulink®, and Stateflow® (optional) to create models using Simulink blocks and Stateflow diagrams. After creating your model, you can run simulations in nonreal-time.

xPC Target allows you to add I/O blocks to your model, and then use the host PC with Real-Time Workshop®, Stateflow Coder (optional) and a C compiler to create executable code. The executable code is download from the host PC to the target PC running the xPC Target real-time kernel. After downloading the executable code, you can run and test your target application in real-time.

2

Advanced Procedures

After learning the basic procedures for creating and running a target application, signal acquisition and parameter tuning, you can try some od the special and advanced procedures with xPC Target.

This chapter includes the following sections:

- **"I/O Driver Blocks"** — Adding I/O driver blocks to your Simulink model connects your model to sensors and actuators.
- **"xPC Target Scope Blocks"** — Adding xPC Target scopes blocks to your Simulink model eliminates the need to create and define scopes after the build process.
- **"Target PC Command Line Interface"** — Enter commands on the target PC for stand alone applications not connected to the host PC.

I/O Driver Blocks

You add I/O driver blocks to your Simulink model to connect your model to physical I/O boards. These I/O boards then connect to the sensors and actuators in the physical system.

This section includes the following topics:

- "xPC Target I/O Driver Blocks"
- "Adding I/O Blocks with the xPC Target Library"
- "Adding I/O Blocks with the Simulink Library Browser"
- "Defining I/O Block Parameters"

xPC Target I/O Driver Blocks

A driver block does not represent an entire board, but an I/O section supported by a board. Therefore, the xPC Target library may have more then one block for each physical board. I/O driver blocks are written as C-code S-functions (noninlined S-functions). We include the source code for the C-code S-functions with xPC Target.

xPC Target supports PCI and ISA busses. If the bus type is not indicated in the driver block number, you can determine the bus type of a driver block by checking the blocks parameter dialog box. The last parameter is either a PCI slot for PCI boards, or a Base Address for ISA boards.

Adding I/O Blocks with the xPC Target Library

xPC Target contains an I/O driver library with Simulink blocks. You can drag-and-drop these blocks from the library to your Simulink model. Alternately, you can access the I/O driver library with the Simulink Library Browser. See “Adding I/O Blocks with the Simulink Library Browser” on page 2-7.

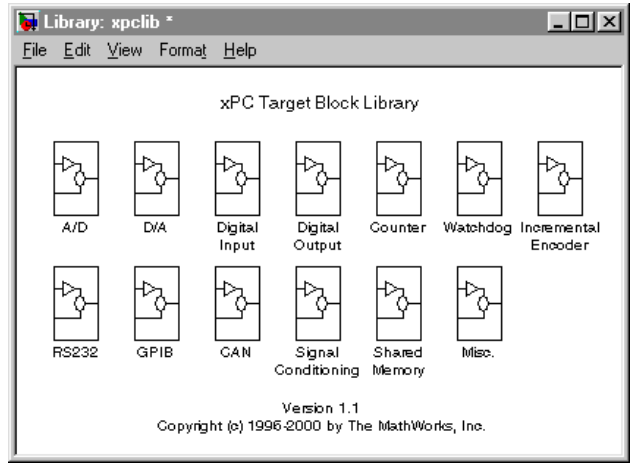
The highest hierarchy level in the library is grouped by I/O function. I/O functions include A/D, D/A, Digital In, Digital Out, Counter, Watchdog, Incremental Encoder, RS232, CAN, GPIB, and shared memory. The second level is grouped by board manufacturer. The manufacturer groups within this second level contain the specific boards.

This procedure uses the Simulink model `xpcosc.mdl` as an example of how to connect an I/O block.

1 In the MATLAB window, type

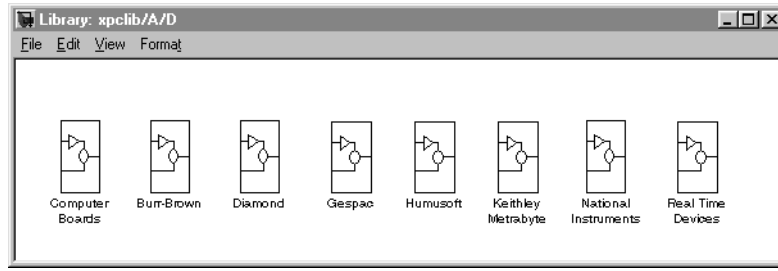
```
xpclib
```

The Library: xpclib window opens.



2 Open a function group. For example, to open the A/D group, double-click the **A/D** block.

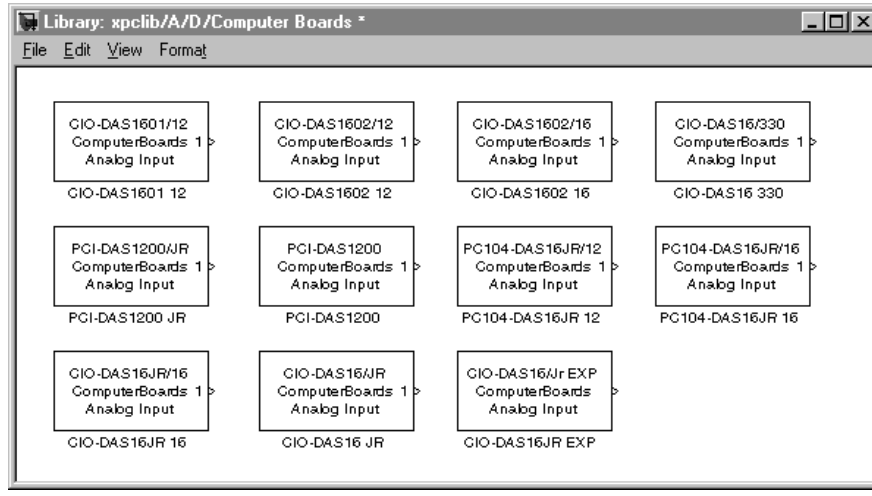
The manufacturer level opens.



Within each manufacturer group are the blocks for a single function.

3 Open a manufacturer group. For example, to open the A/D driver blocks from ComputerBoards, double-click the group marked **ComputerBoards**.

The window with the A/D driver blocks for ComputerBoards opens.



4 In the Simulink window, type

```
xpcosc
```

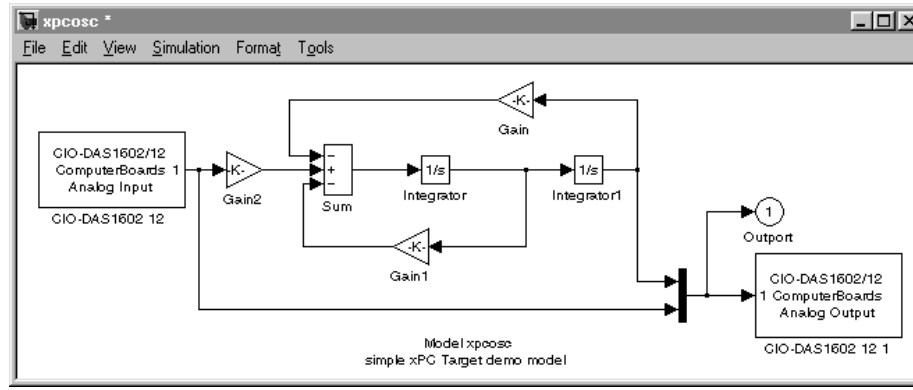
The Simulink block diagram opens for the model `xpcosc.mdl`.

5 From the block library, click-and-drag the name of an A/D board to the Simulink block diagram. Likewise, click-and-drag the name of a D/A board to your model.

Simulink adds the new I/O blocks to your model.

6 Remove the signal generator block and add the analog input block in its place. Remove the scope block and add the analog output block in its place.

The demo model xpcosc should look like the figure shown below.



Your next task is to define the I/O block parameters. See “Defining I/O Block Parameters” on page 2-10.

Adding I/O Blocks with the Simulink Library Browser

You can access the xPC Target driver blocks using the Simulink Library Browser. You drag-and-drop these blocks from the library to your Simulink model. Alternately, you can access driver blocks using the xPC Target I/O driver library, See "Adding I/O Blocks with the xPC Target Library" on page 2-4.

The highest hierarchy level in the library is grouped by I/O function. I/O functions include A/D, D/A, Digital In, Digital Out, Counter, Watchdog, Incremental Encoder, RS232, CAN, GPIB, and shared memory. The second level is grouped by board manufacturer. The manufacturer groups within this second level contain the specific boards.

This procedure uses the Simulink model `xpcosc.mdl` as an example of how to connect an I/O block.

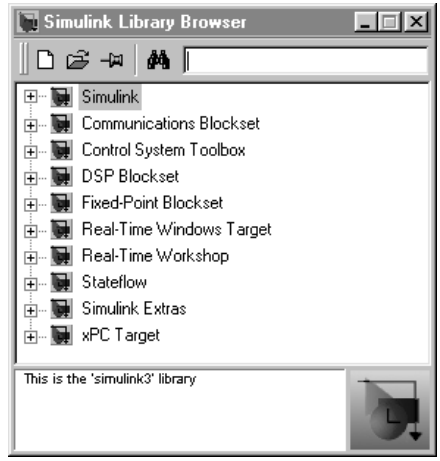
1. In the MATLAB window type

   ```
   xpcosc
   ```

   The Simulink block diagram opens for the model `xpcosc.mdl`.

2. In the Simulink window, and from the **View** menu, click **Show Library Browser**.

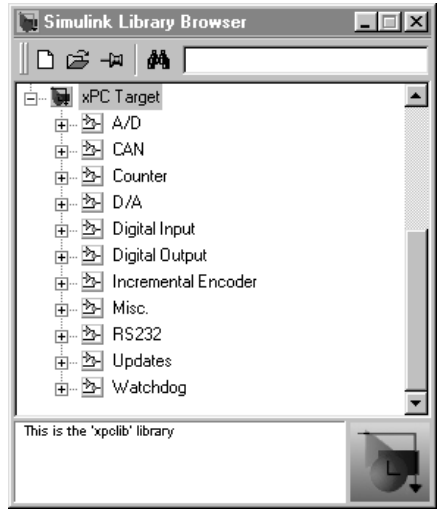
   The Simulink Library Browser window opens. Alternately, you can open the Simulink Library Browser by typing `simulink` in the MATLAB command window.



You can access the xPC Target I/O library by right-clicking **xPC Target**, and then clicking **Open the xPC Target Library**.

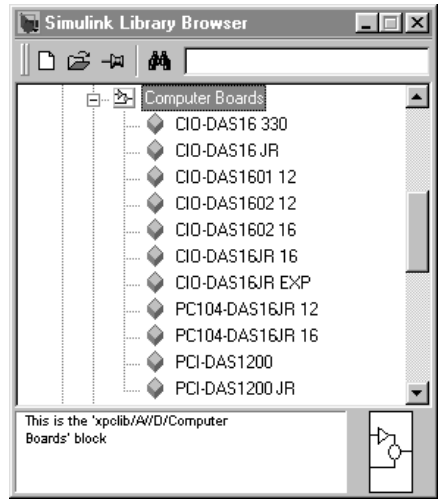
3 Double-click **xPC Target**.

A list of I/O functions opens.



4 Open a function group. For example, to open the A/D group for ComputerBoards, double-click **A/D**, and then double-click **ComputerBoards**.

A list with the A/D driver blocks for ComputerBoards opens.

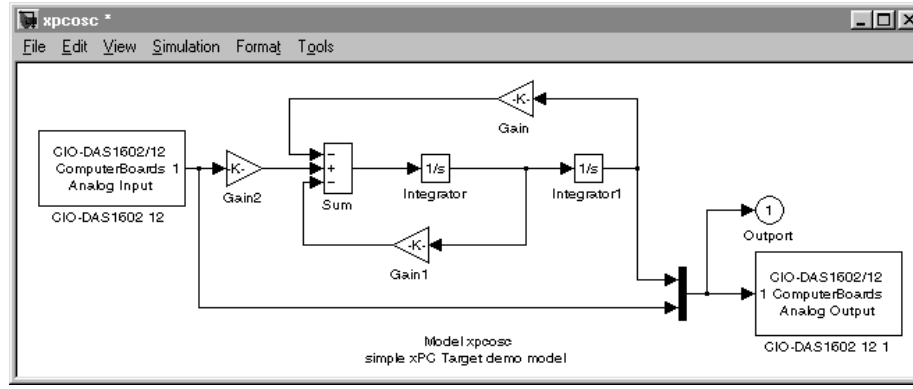


5 From the block library, click-and-drag the name of an A/D board to the Simulink block diagram. Likewise, click-and-drag the name of a D/A board to your model

Simulink adds the new I/O blocks to your model.

6 Remove the signal generator block and add the analog input block in its place. Remove the scope block and add the analog output block in its place.

The model xpcosc should look like the figure shown below.



Your next task is to define the I/O block parameters. See “Defining I/O Block Parameters”.

Defining I/O Block Parameters

The I/O block parameters define values for your physical I/O boards. For example, I/O block parameters include channel numbers for multichannel boards, input and output voltage ranges, and sample time.

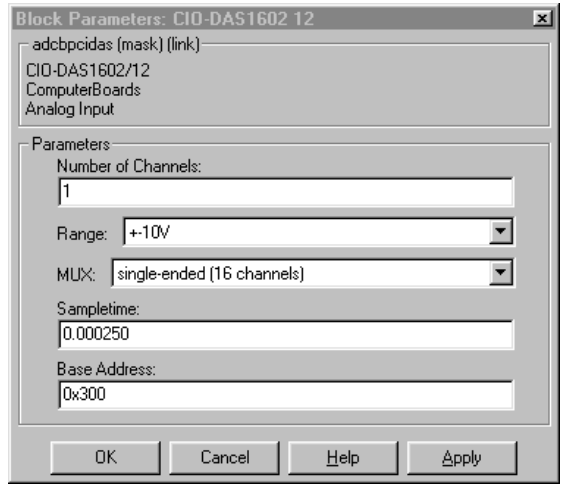
This procedure uses the Simulink model `xpcosc.mdl` as an example, and assumes you have added an analog input and an analog output block to your model. To add an I/O block, see either “Adding I/O Blocks with the xPC Target Library” on page 2-4 or“Adding I/O Blocks with the Simulink Library Browser” on page 2-7.

1 In the Simulink window, double-click the input block labeled **Analog Input**.

  The dialog box for the A/D converter opens.

2 Fill in the dialog box. For example, for a single channel enter `1` in the Number of Channels box, choose `±10 V for the` input range, and choose `single-ended (16 channels)for the` MUX-switch position. Enter the same sample time you entered for the step size in the Simulation Parameters dialog box. Enter the base address for this ISA-bus board.

  The **Block Parameters** dialog box should look similar to the figure shown below.

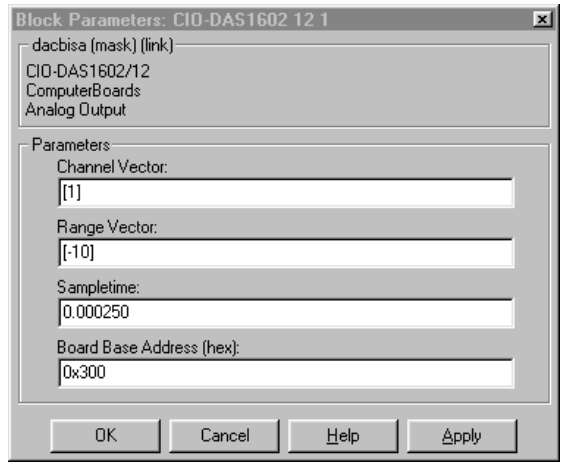


3 In the Simulink window, double-click the output block labeled **Analog Output**.

The dialog box for the D/A converter opens.

4 Fill in the dialog box. For example, for one channel enter [1] in the Channel Vector box, for an ouptut level of ±10 V enter the code [-10] in the Range Vector box. Enter the same sample time you entered for the step size in the Simulation Parameters dialog box. Enter the base address for this ISA-bus board.

The **Block Parameters** dialog box should look similar to the figure shown below.



If you change the sample time by changing the target object property `SampleTime`, the step size you entered in the **Simulation Parameters** dialog box, as well as the sample times you entered in both of the I/O blocks are set to the new value.

Your next task is to build and run the target application, see “Creating the Target Application” on page 3-7.

xPC Target Scope Blocks

Usually scope objects are defined using xPC Target functions or the graphical interface after downloading your target application. An alternative is for you to add special xPC Target scope blocks to you Simulink model. These blocks should not be confused with standard Simulink scope blocks. The xPC Target scope blocks have unique capabilities when used with xPC Target.

This section includes the following topics:

- "xPC Target Scope Blocks"
- "Adding xPC Target Scope Blocks"
- "Defining xPC Target Scope Block Parameters"

xPC Target Scope Blocks

An xPC Target scope block is added to your model the same way you add any Simulink block. After adding an xPC Target scope block, you define the properties of the scope object and the signals you want to display. When the target application is downloaded to the target PC, the a scope is automatically created on the target PC. No additional definitions are necessary.

Note The scope object created on the host PC is not assigned to a MATLAB variable. To assign the scope object, use the target object function `getscope`. Also, if you use the function `remscope` to remove a scope created during the build and download process, and then you restart the target application, the scope is recreated.

Alternative methods for creating xPC Target scopes — For information on using xPC Target functions to create scopes, see "Signal Tracing with MATLAB Commands" on page 3-35. For information on using the xPC Target graphical interface to create scopes, see "Signal Logging with xPC Target Graphical Interface" on page 3-20.

Adding xPC Target Scope Blocks

Adding xPC Target scope blocks to your Simulink model saves you the time to define and select signals after you download the target application to the target PC, and the information is saved with your model.

You can drag an xPC Target scope block into any Simulink model, and the input to the scope can be connected to any block output. If you want to trace more than one signal, add a multiplexer block to your model, and connect the scope block to the multiplexer output.

The following procedure uses the Simulink model `xpcosc.mdl` as an example to show how to connect an xPC Target scope block to your model.

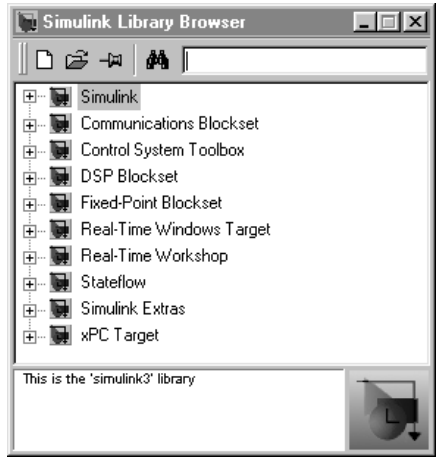
1 In the MATLAB window type

```
xpcosc
```

The Simulink block diagram opens for the model `xpcosc.mdl`.

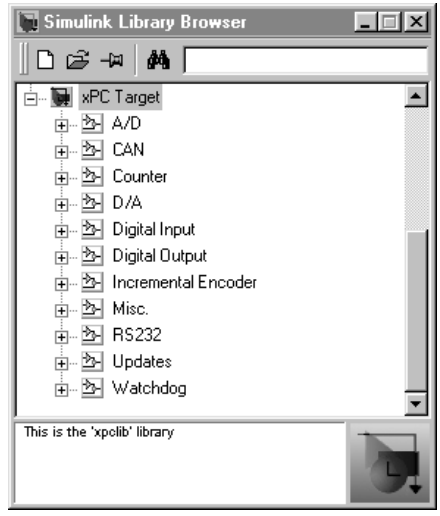
2 In the Simulink window, and from the **View** menu, click **Show Library Browser**.

The Simulink Library Browser window opens.



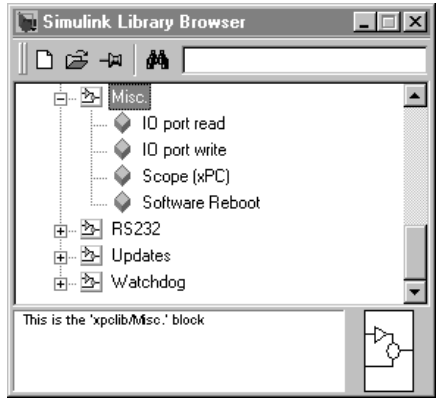
3 Double-click **xPC Target**.

A list of I/O functions opens.



4 Double-click **Misc.**

A list of miscellaneous group blocks opens.

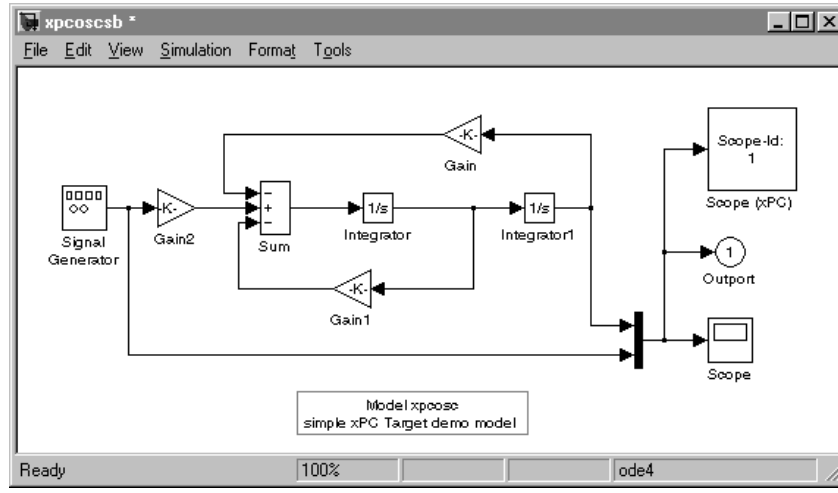


5 Click-and-drag **Scope (xPC)** to your Simulink block diagram.

Simulink adds a new scope block to your model with a scope identifier of 1.

6 Connect the xPC Target scope block with the Simulink scope block.

The model xpcosc should look like the figure shown below.



Your next task is to define the xPC Target scope block parameters. See "Defining xPC Target Scope Block Parameters".

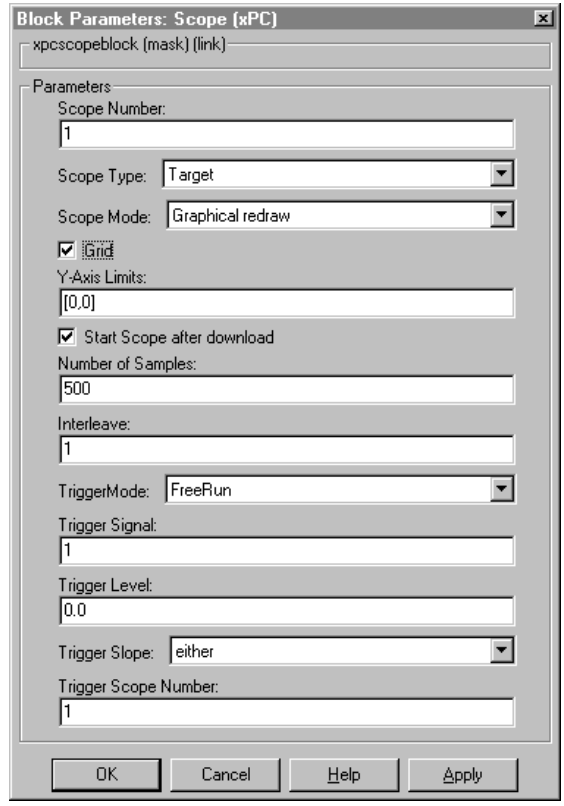
Defining xPC Target Scope Block Parameters

Scope block parameters define the signals to trace on the scope, trigger modes, and the axis range.

This procedure uses the Simulink model `xpcosc.mdl` as an example, and assumes you have added an xPC Target scope block to your model. To add a scope block, see "Adding xPC Target Scope Blocks" on page 2-13.

1 Double-click the scope block labeled **Scope (xPC)**.

The **Block Parameters** dialog box for the scope block opens.



2 In the **Scope Number** box, enter a unique number to identify the scope. This number is used to identify the xPC Target scope block and the scope screen on the host or target computers.

3 From the **Scope Type** list, choose either **Host** or **Target**.

4 From the **Scope Mode** list, choose either **Numerical**, **Graphical (redraw)**, **Graphical (sliding)**, or **Graphical (rolling)**.

If you selected **Host** for the Scope Type, then you can only choose either **Numerical** or **Graphical (redraw)** for the Scope Mode.

5 Select the **Grid** check box to display grid lines on the scope.

6 In the **Y-axis Limits** box, enter a row vector with two elements where the first element is the lower limit of the y-axis and the second element is the upper limit. If you enter `0` for both elements, then the scaling is set to `auto`.

7 Select the **Start Scope after download** check box, to start a scope when the target application is downloaded. With a target scope, the scope window open automatically. With a host scope, you need to open the window by typing `xpcscope`.

8 In the **Number of Samples** box, enter the number of values acquired in a data package before redrawing the graph. In the **Interleave** box, enter a value to collect data at each sample time (`1`) or to collect data at less than every sample time (`2` or greater).

9 From the **Target Mode** list, choose either **FreeRun**, **Software Trigger**, **Signal Trigger**, or **Scope Trigger**.

   If you selected **Signal Trigger**, then in the **Trigger Signal** box, enter the index of a signal. In the **Trigger Level** text box, enter a value for the signal to cross before triggering. And from the **Trigger Slope** list choose **either**, **rising**, or **falling**.

   If you choose **Scope Trigger**, then in the **Trigger Scope Number** box, enter the block number of a scope block. If you use this feature, you must also add a second scope block to your Simulink model.

10 Click **OK**.

Your next task is to build and run the target application. As soon as the target application is built and downloaded, a scope and scope object are automatically created. However, the scope object is not assigned to a MATLAB variable.

Target PC Command Line Interface

You can interact with the xPC Target environment through the target PC command window. This interface is useful with stand-alone applications that are not connected to the host PC.

This section includes the following topics:

- "Using Methods and Properties on the Target PC"
- "Target Object Methods"
- "Target Object Properties"
- "Scope Object Methods"
- "Scope Object Properties"
- "Using Variables on the Target PC"
- "Variable Commands"

Using Methods and Properties on the Target PC

xPC Target uses an object oriented environment on the host PC with methods and properties. While the target PC does not use the same objects, many of the methods on the host PC have equivalent target PC commands. The target PC commands are case sensitive, but the arguments are not case sensitive.

After you have created and downloaded a target application to the target PC, you can use the target PC commands to run and test your application.

1 On the target PC, press C or move the mouse over the command window.

   The target PC command window is activated, and a command line opens. If the command window is already activated, you do not have to press C. In this case, pressing C is taken as the first letter in a command.

2 In the **Cmd** box, type a target PC command. For example, to start your target application, type

```
start <enter>
```

Once the command window is active, you do not have to reactivate it before typing the next command. For a list of target PC commands, see "Target Object Methods" on page 2-20, "Target Object Properties" on page 2-20, "Scope Object Methods" on page 2-21, and "Scope Object Properties" on page 2-22.

Target Object Methods

When using the target PC command line interface, target object methods are limited to starting and stopping the target application.

The following table lists the syntax for the target commands that you can use on the target PC. The MATLAB equivalent syntax is shown in the right column, and the target object name `tg` is used as an example for the MATLAB methods.

Target PC	MATLAB
`start`	`tg.start` or `+tg`
`stop`	`tg.stop` or `-tg`
reboot	tg.reboot

Target Object Properties

When using the target PC command line interface, target object properties are limited to parameters, signals, stoptime, and sampletime. Notice the difference between a parameter index (0, 1, . . .) and a parameter name (P0, P1, . . .).

The following table lists the syntax for the target commands that you can use on the target PC. The MATLAB equivalent syntax is shown in the right column, and the target object name `tg` is used as an example for the MATLAB methods.

Target PC	MATLAB
`setpar` *`parameter_index`* `=` *`number`*	`set(tg, '`*`parameter_name`*`',` *`number`*`)`
`getpar` *`parameter_index`*	`get(tg, '`*`parameter_name`*`')`
`stoptime =` *`number`*	`tg.stoptime =` *`number`*
`sampletime =` *`number`*	`tg.sampletime =` *`number`* `set(tg, 'SampleTime',` *`number`*`)`

Target PC	MATLAB
parameter_name (P0, P1, . . .) *parameter_name* = *number*	tg.*parameter_name* tg.*parameter_name* = *number*
signal_name (S0, S1, . . .)	tg.S#

Scope Object Methods

When using the target PC command line interface, you use scope object methods to start a scope, and add signal traces. Notice the methods addscope and remscope are target object methods on the host PC, and notice the difference between a signal index (0, 1, . . .) and a signal name (S0, S1, . . .)

The following table lists the syntax for the target commands that you can use on the target PC. The MATLAB equivalent syntax is shown in the right column. The target object name tg and the scope object name sc is used as an example for the MATLAB methods.

Target PC	MATLAB
addscope *scope_index* addscope	tg.addscope(*scope_index*) tg.addscope
remscope *scope_index* remscope 'all'	tg.remscope(*scope_index*) tg.remscope
startscope *scope_index*	sc.start or +sc
stopscope *scope_index*	sc.stop or -sc
addsignal *scope_index* = *signal_index1*, *signal_index2*, . . .	sc.addsignal(*signal_index_vector*)
remsignal *scope_index* = *signal_index1*, *signal_index2*, . . .	sc.remsignal(*signal_index_vector*)

Target PC	MATLAB
`viewmode` *scope_index* or left click the scope window `viewmode all` or right click any scope window Press function key for scope, and then press V to toggle viewmode	
`ylimit` *scope_index* `ylimit` *scope_index* = auto `ylimit` *scope_index* = *num1*, *num2*	
`grid` *scope_index* `on` `grid` *scope_index* `off`	

Scope Object Properties

When using the target PC command line interface, scope object properties are limited to those shown in the following table. Notice the difference between a scope index (0, 1, . . .) and the MATLAB variable name for the scope object on the host PC. The scope index is indicated in the top left corner of a scope window (SC0, SC1, . . .).

If a scope is running, you need to stop the scope before you can change a scope property.

The following table lists the syntax for the target commands that you can use on the target PC. The MATLAB equivalent syntax is shown in the right column, and the scope object name sc is used as an example for the MATLAB methods

Target PC	MATLAB
`numsamples` *scope_index* = *number*	`sc.NumSamples =` *number*
`decimation` *scope_index* = *number*	`sc.Decimation =` *number*

Target PC	MATLAB
scopemode *scope_index* = 0 or numerical, 1 or redraw, 2 or sliding, 3 or rolling	sc.Mode = 'numerical', 'redraw', 'sliding', 'rolling'
triggermode *scope_index* = 0, freerun, 1 software, 2, signal, 3, scope	sc.TriggerMode = 'freerun', 'software', 'signal', 'scope'
numprepostsamples *scope_index* = *number*	sc.NumPrePostSamples = *number*
triggersignal *scope_index* = *signal_index*	sc.TriggerSignal = *signal_index*
triggerlever *scope_index* = *number*	sc.TriggerLevel = *number*
triggerslope *scope_index* = 0, either, 1, rising, 2, falling	sc.TriggerSlope = 'Either', 'Rising', 'Falling'
triggerscope *scope_index2* = *scope_index1*	sc.TriggerScope = *scope_index1*
Press function key for scope, and then press S or move mouse into the socpe window	sc.trigger

Using Variables on the Target PC

Use variables to tag unfamiliar commands, parameter indices, and signal indexes with more descriptive names.

After you have created and downloaded a target application to the target PC, you can create target PC variables.

1 On the target PC, press C.

   The target PC command window is activated, and a command line opens.

2 In the **Cmd** box, type a variable command. For example, if you have a parameter that controls a motor, you could create the variables `on` and `off` by typing

```
setvar on = p7= 1
setvar off = p7=0
```

3 Type a variable name. For example, to turn the motor on, type

```
on
```

   The parameter P7 is changed to `1`, and the motor turns on.

Variable Commands

The following table lists the syntax for the target commands that you can use on the target PC. The MATLAB equivalent syntax is shown in the right column.

Target PC	MATLAB
`setvar` *`variable_name = target command`*	
`getvar` *`variable_name`*	
`delvar` *`variable_name`*	
`delallvar`	

3

Network Communication

xPC Target provides support for TCP/IP and UDP/IP communication protocols.

This chapter includes the following sections:

- **"Web Interface"** — Connect a target application to any computer connected to the network.
- **User Datagram Protocol (UDP)** — Send and receive messages from a target application using UDP packets.

Web Interface

xPC Target has a Web server build in the TCP/IP mode to the kernel that allows you to interact your target application using a Web browser. If the target PC is connected to a network, you can use a Web browser to interact with the target application from any computer connected to a network

Currently this feature is limited to Microsoft Internet Explorer (version 4.0 or later) and Netscape Navigator (version 4.5 or later) are the supported browsers.

This section includes the following sections:

- **Connecting the Web Interface through TCP/IP**
- **Connecting the Web Interface through RS232**
- **Using the Main Page**
- **Changing WWW Properties**
- **Viewing Signals with the Web Browser**
- **Using Scopes with the Web Browser**
- **Viewing and Changing Parameters with the Web Interface**
- **Changing Access Levels to the Web Browser**

Connecting the Web Interface through TCP/IP

If your host PC and target PC are connected with a network cable, you can connect the target application on the target PC to a Web browser on the host PC.

The TCP/IP stack on the xPC Target kernel supports only one simultaneous connection, since its main objective is real-time applications. This connection is shared between MATLAB and the Web browser. This also means that only one browser or MATLAB is able to connect at one time.

Before you connect your Web browser on the host PC, you must load a target application onto the target PC. The target application does not have to be running, but it must be loaded. Also, your browser must have JavaScript and StyleSheets turned on.

1 In the MATLAB window, type

```
xpcwwwenable
```

MATLAB is disconnected from the target PC, and the connection is reset for connecting to another client. If you do not use this command, your Web browser may not be able to connect to the target PC.

2 Open a Web browser. In the address box, enter the IP address and port number you entered in the xPC Target Setup window. For example, if the target computer IP address is 192.168.0.1 and the port is 22222, type

```
http://192.168.0.1:22222/
```

The browser loads the xPC Target Web interface frame and pages.

Connecting the Web Interface through RS232

If the host PC and target PC are connected with a serial cable, instead of a network cable, you can still connect the target application on the target PC to a Web browser on the host PC. xPC Target includes a TCP/IP to RS232 mapping application. This application writes whatever it receives from the RS232 connection to a TCP/IP port and vice versa.

Before you connect your Web browser on the host PC, you must load a target application onto the target PC. The target application does not have to be running, but it must be loaded. Also, your Web browser must have JavaScript and StyleSheets turned on.

1 In the MATLAB window, type

```
xpcwwwenable or close(xpc)
```

MATLAB is disconnected from the target PC leaving the target PC ready to connect to another client. The TCP/IP stack on the xPC Target kernel supports only one simultaneous connection. If you do not use this command, the TCP/IP to RS232 gateway may not be able to connect to the target PC.

2 Open a DOS command window, and enter the command to start the TCP/IP to RS232 gateway. For example, if the target PC is connected to COM1 and you would like to use the TCP/IP port 222222, type the following

```
c:\MATLABR12\toolbox\rtw\targets\xpc\xpc\bin\xpctcp2ser -v -t
22222 -c 1
```

The TCP/IP to RS232 gateway starts running, and the DOS command window displays the message

```
*-------------------------------------------------------------*
*             xPC Target TCP/IP to RS232 gateway              *
*               Copyright 2001 The MathWorks                  *
*-------------------------------------------------------------*
Connecting COM to TCP port 22222
Waiting to connect
```

If you did not close the MATLAB to target application connection, then DOS displays the message, `Could not initialize COM port`.

3 Open a Web browser. In the address box, enter

```
http://localhost:22222/
```

The Web browser loads the xPC Target Web interface pages.

4 Using the Web interface, start and stop the target application, add scopes add signals, and change parameters.

5 In the DOS command window, press Ctrl+C

The TCP/IP to RS232 Gateway stops running, and the DOS command window displays the message

```
Interrupt received, shutting down
```

The gateway application has a handler that responds to Ctrl-C by disconnecting and shutting down cleanly. In this case, Ctrl-C is not used to abort the application.

6 In the MATLAB command window, type

```
xpc
```

MATLAB reconnects to the target application and lists the properties of the target object.

If you did not close the gateway application, then MATLAB displays the message

```
Error in ==>
C:\MATLABR12\toolbox\rtw\targets\xpc\xpc\@xpc\xpc.m
On line 31  ==> sync(xpcObj);
```

To correct this problem, you must close MATLAB and then restart it.

Syntax for the xpctcp2ser Command

The syntax for the xpctcp2ser command is

```
xpctcp2ser [-v] [-n] [-t tcpPort] [-c comPort]
```

The options are described in the following table.

Command line option	Description
-v	Produces a line of output every time a client connects or disconnects.
-n	Allows non-local connections. By default, only clients from the same computer that the gateway is running on are allowed to connect. This option allows anybody to connect to the gateway. If you don't use this option, only the host PC which is connected to the target PC with a serial cable will be able to connect to the selected port. For example, if you start the gateway on your host PC, with the default ports, you can type in the web browser http://localhost:22222/ . However, if you try to connect to http://DomainName.com:22222/ , you will probably get a connection error.

Command line option	Description
-t tcpPort	Uses TCP port tcpPort. Default t is 22222. For example, to connect to port 20010, type `-t 20010`.
-h	Prints a help message.
-c comPort	Uses COM port comPort (1 <= comPort <= 4), Default is 1. For example, to use COM2, type `-c 2`

7 In the MATLAB window, type

```
!c:\MATLABR12\toolbox\rtw\targets\xpc\xpc\bin\xpctcp2ser -h
```

Alternatively, you can open a DOS command window and enter

```
c:\MATLABR12\toolbox\rtw\targets\xpc\xpc\bin\xpctcp2ser -h
```

MATLAB displays the following

```
Usage:
xpctcp2ser [-v] [-n] [-t tcpPort] [-c comPort]
xpctcp2ser -h

 The command line options are as follows:
 -v         : Verbose output.
 -c comPort : Use Com Port comPort (1 <= comPort <= 4),
              defaults to 1
 -t tcpPort : Use TCP port # tcpPort, defaults to 22222
 -h         : Print a help message.
 -n         : Allow Non-local connections. By default,
              only clients from the same computer that the
              gateway is running on are allowed to connect.
              This option allows anybody to connect to the
              gateway.
```

Using the Main Page

The Main page is divided into four parts, one below the other. The four parts are: System Status, xPC Target Properties, Navigation, and WWW Properties.

After you connect a Web browser to the target PC you can use the Main page to control the target application.

1 In the left frame, click the **Refresh** button.

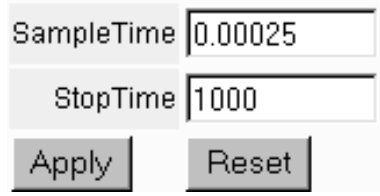
System status information in the top cell is uploaded from the target PC. If the right frame is either the Signals List page or the Screen Shot page, updating the left frame also updates the right frame.

2 Click the **Start Execution** button.

The target application begins running on the target PC, the Status line is changed from Stopped to Running, and the **Start Execution** button text changes to **Stop Execution**.

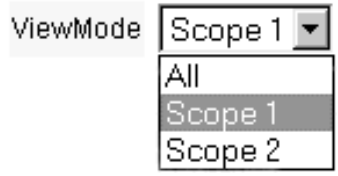
3 Update the execution time and average task execution time (TET). Click the **Refresh** button. To stop the target application, click the **Stop Execution** button.

4 Enter new values in the **Stop Time** and **Sample Time** boxes, and then click the **Apply** button. You can enter `-1` or `Inf` in the Stop Time box for an infinite stop time.



The new property values are downloaded to the target application. Notice, the sample time box is visible only when the target application is stopped. You cannot change the sample time while a target application is running.

5 Select scopes to view on the target PC. From the **ViewMode** list choose one or all of the scopes to view.



Note the **ViewMode** button is visible only if you add two or more scopes to the target PC.

Changing WWW Properties

The WWW Properties cell in the left frame contains fields that affect the display on the Web interface itself, and not the application. There are two fields: maximum signal width to display, and refresh interval.

1 In the **Maximum Signal Width** enter `-1`, `Inf` (all signals), `1` (show only scalar signals), `2` (show scalar and vector signals less than or equal to 2 wide), or `n` (show signals with a width less than `n`)

Signals with a width greater than the value you enter, are not displayed on the Signals page.

2 In the **Refresh Interval** box, enter a value greater than 10. For example, enter 20.

   The signal page updates automatically every `20` seconds. Entering `-1` or `Inf` does not automatically refresh the page.

Sometimes, both of the frames try to update simultaneously, or the auto refresh starts before the previous load has finished. This problem may happen with slow network connections. In this case, increase the refresh interval or manually refresh the browser (Set the Refresh Interval = Inf).

This may also happen when you are trying to update a parameter or property at the same time as the page is automatically refreshing.

Sometimes, when a race condition occurs, the browser becomes confused about the format, and you may have to refresh it. This should not happen too often.

Viewing Signals with the Web Browser

The Signal page is a list of the signals in your model.

After you connect a Web browser to the target PC you can use the Signals page to view signal data.

1 In the left frame, click the **Signals** button.

   The Signals page is loaded in the right frame with a list of signals and the current values.

2 On the Signals page in the right frame, click the **Refresh** button.

   The Signals page is updated with the current values. Vector/matrix signals are expanded and indexed in the same column-major format that MATLAB uses. This may be affected by the Maximum Signal Width value you enter in the left frame.

3 In the left frame, click the **Screen Shot** button.

   The Screen Shot page is loaded and a copy of the current target PC screen is displayed. The screen shot uses the Portable Network Graphics file format `PNG`.

Using Scopes with the Web Browser

The Web browser interface allows you to visualize data using an interactive graphical interface.

After you connect a Web browser to the target PC you can use the Scopes page to add, remove and control scopes on the target PC.

1 In the left frame, click the **Scopes** button.

   The Scopes List page is loaded into the right frame.

2 Click the **Add Scope** button.

   A scope of type target is created and displayed on the target PC. The scopes page displays a list of all the scopes present. The capability exists to add a new scope, remove existing scopes, and control all aspects of a scope from this page.

   Note If any host scopes exist, they will be visible and controllable from this page. They may even be removed. However, there is no capability to add any scopes of type host from the Scope page.

3 Click the **Edit** button.

   The scope editing page opens. From this page, you can edit the properties of any scope, and control the scope.

4 Click the **Add Signals** button.

   The browser displays an **Add New Signals** list.

5 Select the check boxes next to the signal names, and then click the **Apply** button.

   A **Remove Existing Signals** list is added above the **Add New Signals** list.

You do not have to stop a scope to make changes. If stopping the scope is necessary, the Web interface stops it automatically and then restarts it if necessary when the changes are made. It will not restart the scope if the state was originally stopped.

Viewing and Changing Parameters with the Web Interface

The parameters page displays a list of all the tunable parameters in your model. Row and column indices for vector/matrix parameters are also shown.

After you connect a Web browser to the target PC you can use the Parameters page to change parameters in your target application while it is running in real time.

1 In the left frame, click the **Parameters** button.

   The Parameter List page is loaded into the right frame.

   If the parameter is a scalar parameter, the present parameter value is shown in a box that you can edit.

   If the parameter is a vector/matrix, there is a button which takes you to another page which displays the vector/matrix (in the correct shape) and enables you to edit the parameter

2 Enter a new parameter value into one or more of the parameter boxes, and then click the **Apply** button.

   The new parameter values are uploaded to the target application.

Changing Access Levels to the Web Browser

The Web browser interface allows you to set access levels to the target application. The different levels limit access to the target application. The highest level, 0, is the default level and allows full access. The lowest level 4, only allows signal monitoring and tracing with your target application.

1 In the Simulink window, click **Simulation Parameters**. On the Simulation Parameter dialog box, click the **Real-Time Workshop** tab.

   Access levels are set in the **System target file** box. For example, to set the access level to `1`, enter

   ```
   xpctarget.tlc -axpcWWWAccessLevel=1
   ```

   The effect of not specifying -axpcWWWAccessLevel=# is that the access level of 0 (the highest) is set.

2 Click **OK**.

The various fields disappear depending on the access level. For example, if your access level does not allow you access to the parameters, you will not see the button for parameters.

There are various access levels for monitoring, which will allow different levels of hiding. The proposed setup is described below. Each level builds up on the previous one, so only the incremental hiding of each successive level is described

Level 0 — Full access to all pages and functions.

Level 1 — Cannot change the sample and stop times. Cannot change parameters, but can view parameters.

Level 2 — Cannot start and stop execution of the target application.

Level 3 — Cannot view parameters. Cannot add new scopes, but can edit existing scopes.

Level 4 — Cannot edit existing scopes on the Scopes page. Cannot add or remove signals on the Scopes page. Cannot view the Signals page and the Parameters page.

User Datagram Protocol (UDP)

xPC Target supports communication with another system using User Datagram Protocol (UDP) packets. UDP is a transport protocol similar to TCP. However, unlike TCP, UDP provides a direct method to send and receive packets over an IP network. UDP uses this direct method at the expense of reliability by limiting error checking and recovery.

This section includes the following topics:

- What Is UDP?
- Why UDP?
- UDP Communication Setup
- UDP Receive Block
- UDP Send Block
- UDP Pack Block
- UDP Unpack Block
- UDP Example

What Is UDP?

The User Datagram Protocol (UDP) is a transport protocol layered on top of the Internet Protocol (IP) and is commonly known as UDP/IP. It is analogous to TCP/IP. A convenient way to present the details of UDP/IP is by comparison to TCP/IP as presented below:

- **Connection Versus Connectionless** — TCP is a *connection based* protocol, while UDP is a *connectionless* protocol. In TCP, the two ends of the communication link must be connected at all times during the communication. An application using UDP prepares a packet and sends it to the receiver's address without first checking to see if the receiver is ready to receive a packet. If the receiving end is not ready to receive a packet, the packet is lost
- **Stream Versus Packet** — TCP is a *stream-oriented* protocol, while UDP is a *packet-oriented* protocol. This means that TCP is considered to be a long stream of data that is transmitted from one end to the other with another long stream of data flowing in the other direction. The TCP/IP stack is responsible for breaking the stream of data into packets and sending those

packets while the stack at the other end is responsible for reassembling the packets into a data stream using information in the packet headers. UDP, on the other hand, is a packet-oriented protocol where the application itself divides the data into packets and sends them to the other end. The other end does not have to reassemble the data into a stream. Note, some applications may indeed present the data as a stream when the underlying protocol is UDP. However, this is the layering of an additional protocol on top of UDP, and it is not something inherent in the UDP protocol itself.

- **TCP Is a Reliable Protocol, While UDP Is Unreliable** — The packets that are sent by TCP contain a unique sequence number. The starting sequence number is communicated to the other side at the beginning of communication. Also, the receiver acknowledges each packet, and the acknowledgement contains the sequence number so that the sender knows which packet was acknowledged. This implies that any packets lost on the way can be retransmitted (the sender would know that they did not reach their destination since it had not received an acknowledgement). Also, packets that arrive out of sequence can be reassembled in the proper order by the receiver.

  Further, timeouts can be established, since the sender will know (from the first few packets) how long it takes on average for a packet to be sent and its acknowledgment received. UDP, on the other hand, simply sends the packet and does not keep track of them. Thus, if packets arrive out of sequence, or are lost in transmission, the receiving end (or the sending end) has no way of knowing.

TCP communication may be compared to a telephone conversation where a connection is required at all times and two-way streaming data (the words spoken by each party to the conversation) are exchanged. UDP, on the other hand, may be compared to sending letters by mail (without a return address). If the other party is not found, or the letter is lost in transit, it is simply discarded. The analogy fails, however, when considering the speed of communication. Both TCP and UDP communication happen roughly at the same speed since both use the underlying Internet Protocol (IP) layer.

Note *Unreliable* is used in the sense of "not guaranteed to succeed" as opposed to "will fail a lot of the time." In practice, UDP is quite reliable as long as the receiving socket is active, and is processing data as quickly as it arrives.

Why UDP?

UDP was chosen as the transport layer for xPC Target precisely because of its lightweight nature. Since the primary objective of an application running in the xPC Target framework is real-time, the lightweight nature of UDP ensures that the real-time application will have a maximum chance of succeeding in real-time execution. Also, the datagram nature of UDP is ideal for sending samples of data from the Simulink/RTW generated application. Since TCP is stream oriented, separators between sets of data will have to be used for the data to be processed in samples. It is easier to build an application to deal with unreliable data than it is to decode all of this information in real-time. Also, if the application is unable to process the data as quickly as it arrives, the following packets can just be ignored and only the most recent packet can be used.

Communication can involve a packet made up of any Simulink data type (`double`, `int8`, `int32`, `uint8`, etc.), or a combination of these. xPC Target provides blocks for combining various signals into one packet (packing), and then transmitting it. Also, xPC Target provides blocks for splitting a packet (unpacking) into its component signals which can then be used in a Simulink model. The maximum size of a packet is limited to about 500 bytes.

UDP Communication Setup

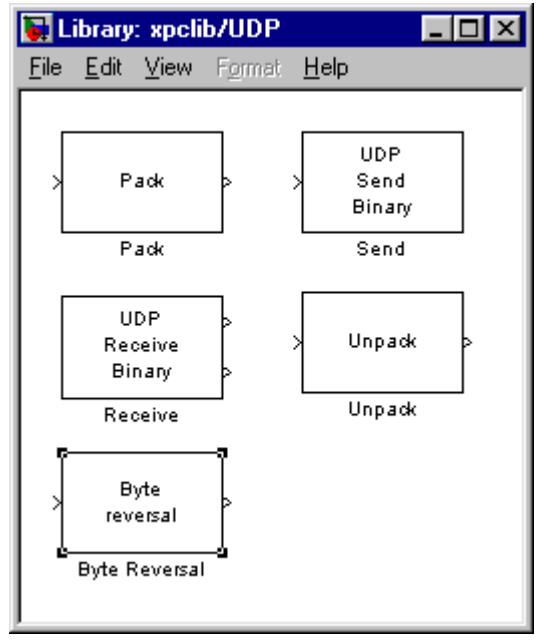
The infrastructure provided in the xPC Target Library for UDP communication consists mainly of two blocks — a Send block and a Receive block. These blocks may be found in the xPC Target Library available from the Simulink Library under **xPC Target**, or you can access then form the MATLAB command line, by typing

```
xpclib
```

The blocks are located under the UDP heading in the library. The Send block takes as input a vector of type `uint8`, which it sends. This is limited to a length of about 500 bytes (i.e., a 1x500 vector). Similarly, the Receive block outputs a

vector of `uint8`. To convert arbitrary Simulink data types into this vector of `uint8`, a Pack block is provided, while an Unpack block is provided to convert a vector of uint8s back into arbitrary Simulink data types. The UDP part of the xPC Target Block Library is shown below.

xPC Target includes a Byte Reversal block for communication with *bit-endian* architecture systems. You do not need this block if you are communicating between PC systems running either xPC Target or Microsoft Windows.

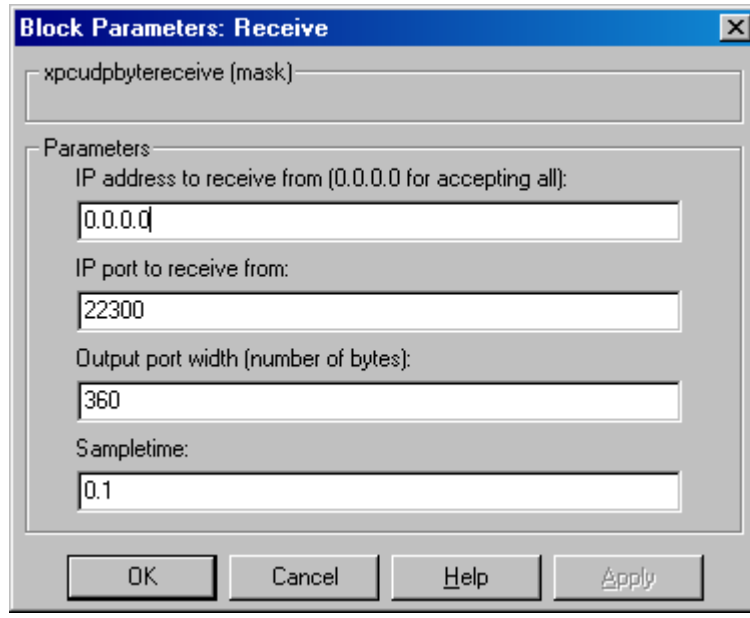


All the blocks are set up to work both from within Simulink and from an application running under xPC Target. However, when using a Simulink simulation and an xPC Target application to communicate, or when using two Simulink models, care must be taken. This is because a Simulink model inherently executes in non-real time and may be several times faster or slower than real time. The sample time of the send and receive blocks and the sample time of the Simulink model must be set so that the communication can proceed properly.

UDP Receive Block

The Receive block has two output ports. The first port is the actual output of the received data as a vector of `uint8` while the second one is a flag indicating whether any new data has been received. This port outputs a value of `1` for the sample when there is new data and a `0`, otherwise. The default behavior of the Receive block is to keep the previous output when there is no new data. This behavior can be modified by the user by using the second port to flag when there is new data.

The Block Parameters for the Review block are shown below.



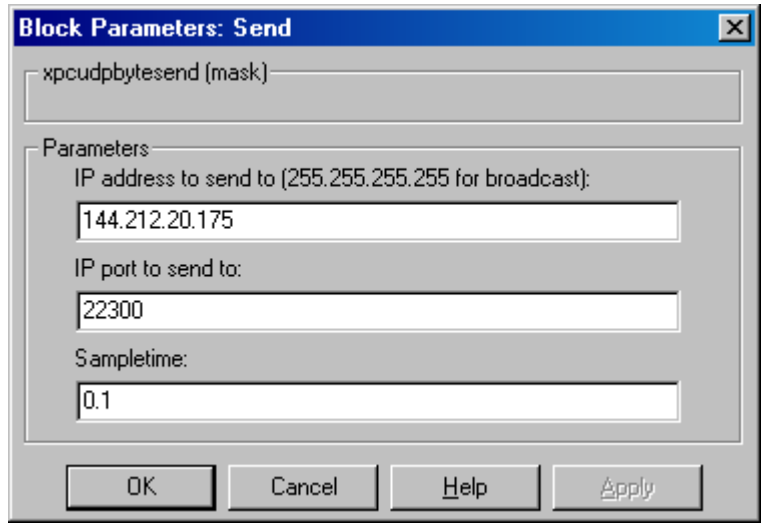
The **IP address to receive from** can be left with the default value of 0.0.0.0. This will accept all UDP packets from any other computer. Otherwise, if set to a specific IP address, only packets arriving from that IP address will be received. The **IP port to receive from** is the port which the block will accept data from. The other end of the communication will have to send data to the port specified here. The output port width is the width of the acceptable packets. This may be obtained when designing the other side (send side) of the communication. The sample time may be set to -1 for inheritable sample time, but it is recommended that this be set to some specific (large) value to eliminate

chances of dropped packets. This is especially true when you are using a small sample time.

UDP Send Block

The Send block has only one input port that receives the `uint8` vector that is sent as a UDP packet.

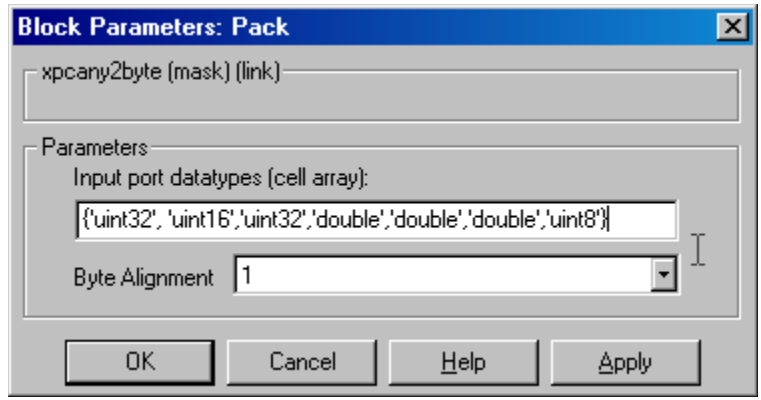
The Block Parameters for the Send block are shown below.



The **IP address to send to** and the **IP port to send to** have to be specified in the appropriate locations. The local IP port that is used for sending will be determined automatically by the networking stack. The sample time should be set to an appropriate value, with similar considerations as in the receive block.

UDP Pack Block

The Pack Block is used to convert one or more Simulink signals of varying data types to a single vector of `uint8` as required by the Send Block. The data types for the different signals must be specified as part of the block parameters while the sizes of each signal are determined automatically.



As seen in the figure above, the data types of each of the signals have to be specified as a cell array of strings in the correct order. Once this is done, the block will automatically convert itself to one with the correct number of input ports. There is always one output port. The supported data types are: `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, and `boolean`. The byte alignment field specifies how the data types are aligned. The possible values are: 1, 2, 4 and 8. The byte alignment scheme is simple, and ensures that each element in the list of signals starts on a boundary specified by the alignment relative to the start of the vector. For example, say the Input port data types are specified as

```
{'uint8','uint32','single','int16','double'}
```

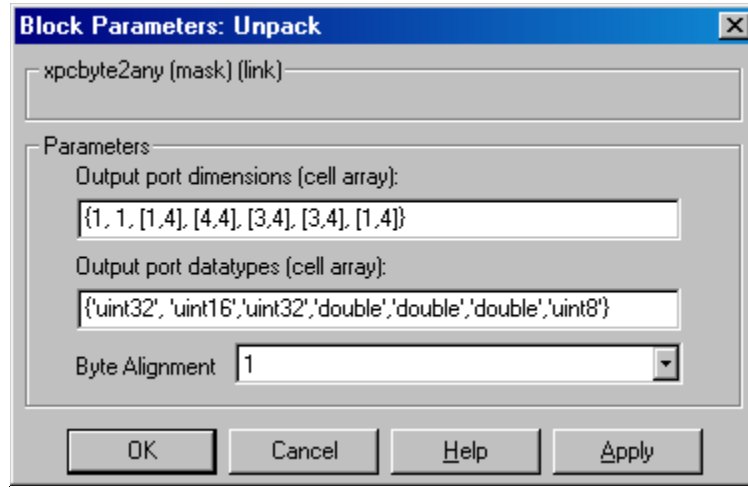
and an alignment of 4 is used. Assume also that all the signals are scalars. The first signal will then start at byte 0 (this is always true), the second at byte 4, the third at byte 8, the fourth at byte 12, and the fifth at byte 16. Note that the sizes of the data types used in this example are 1, 4, 4, 2, and 8 bytes respectively. This implies that there are "holes" of 3 bytes between the first and second signal and 2 bytes between the fourth and fifth signal.

A byte alignment of 1 means the tightest possible packing. That is, there are no holes for any combination of signals and data types.

Note Individual elements of vector/matrix signals are not byte aligned: only the entire vector/matrix is byte aligned. The individual elements are tightly packed with respect to the first element.

UDP Unpack Block

This block is the exact analog of the Pack block. It receives a vector of `uint8` and outputs various Simulink data types in different sizes.



As shown in the figure above, the **Output port datatypes** field is the same as in the **Input port data types** field of the matching Pack block. The Pack block is on the sending side and the Unpack block is on the receiving side in different models. The **Output port dimensions** field contains a cell array, with each element the dimension as returned by the size function in MATLAB of the corresponding signal. This should normally be the same as the dimensions of the signals feeding into the corresponding Pack block.

Note on Byte Alignment

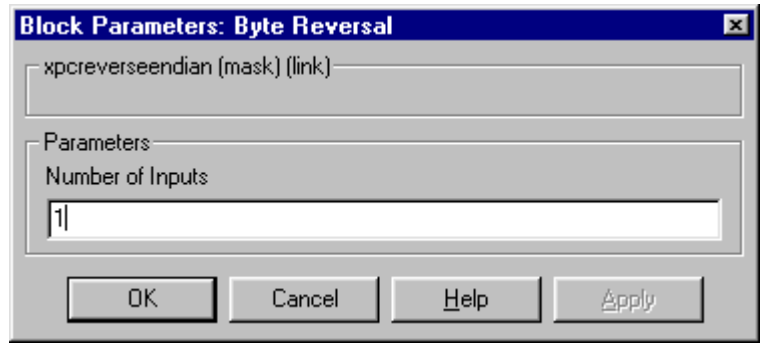
The byte alignment feature provided in the Pack and Unpack blocks is primarily intended for interfacing a system running xPC Target to another system that is running neither Simulink nor xPC Target. For example, the

data on the other end may be in the form of a C `struct`, which is subject to the byte alignment convention of the compiler used. We recommend using a byte alignment value of 1 (tightly packed) whenever possible. This, of course, is easily accomplished when UDP I/O is used to exchange data between two xPC Target systems or between xPC Target and Simulink.

Even when communication is between xPC Target and a system using a C `struct`, the use of compiler pragmas may help to pack the structure tightly. For example, `#pragma pack(1)` is common to several compilers. The byte alignment blocks are provided for the case when this is not possible. We provide an example later for the case of a C structure, where several pack and unpack blocks are used to force the proper alignment of data.

UDP Byte Reversal Block.

You use the Byte Reversal block for communication between an xPC Target system and a system running with a processor that is *big-endian*. Processors compatible with the Intel 80x86 family are always *little-endian*. For this situation, you should insert a Byte Reversal block before the Pack block and just after the Unpack block to ensure that the values are transmitted properly.



This block takes just one parameter, the number of inputs. The number of input ports adjusts automatically to follow this parameter, and the number of outputs is equal to the number of inputs.

UDP Example

In this section, we provide an example of how to set up two way data exchange between two xPC Target systems, between xPC Target and Simulink, or

between two Simulink models. When one or both of the systems is running Simulink in non real-time, care must be taken to set the sample time properly.

Our hypothetical models are called Model A and Model B. Two different sets of data are transferred between these two models. One set from Model A to Model B and another set in the opposite direction.

The data to transfer is in the following order:

Model A to Model B

- `uint8` (3x3)
- `int16` (1x1)
- `double` (2x4)

Model B to Model A

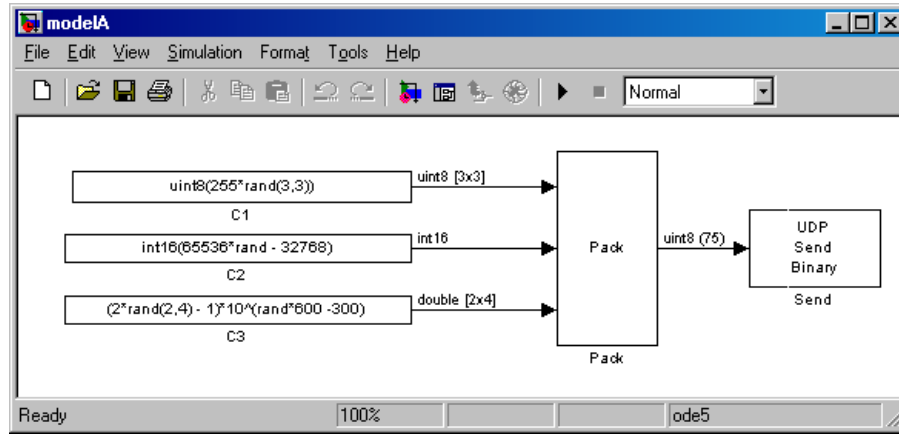
- `single` (4x1)
- `double` (2x2)
- `uint32` (2x2)
- `int8` (5x3)

For the purposes of this example, all the models are generated using Simulink Constant blocks that use the MATLAB random number function (rand). The random numbers are generated by Real Time Workshop using this function at the time of code generation. To generate the vector of `uint8` (3x3), use the MATLAB function.

```
uint8(255 * rand(3,3))
```

since 255 is the maximum value for an unsigned 8-bit integer. The other values are generated similarly.

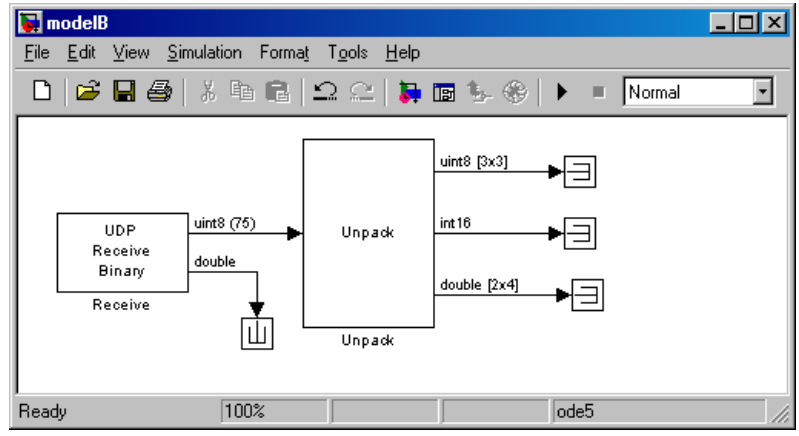
With this setup, construct the send side of modelA.



Note that **Port Data Types** and **Signal dimensions** have been turned on from the **Format** menu, showing us that the width of the UDP packet to be sent is 75 bytes. The parameters used in the Pack block are Input port datatypes `{'uint8','int16','double'}` and Byte Alignment 1.

For the Send block, set the **IP Address to send to** to `192.168.0.2`. This is the hypothetical address of the system that will run Model B. Set the **IP Port to send to** to `25000` (picked arbitrarily). The sample time is set to `0.01`.

Use this information to construct the receive end of Model B.

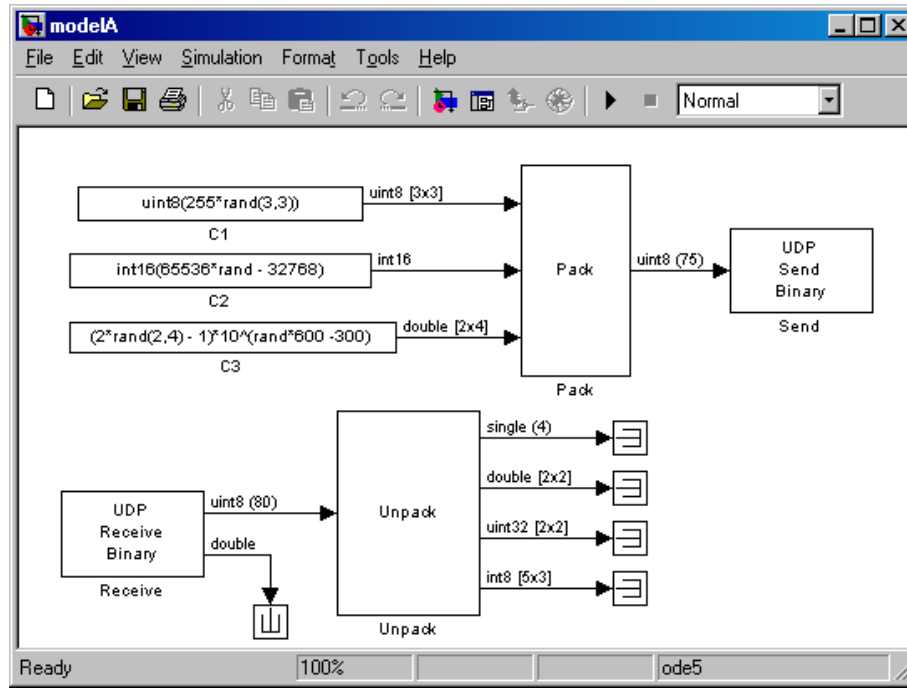


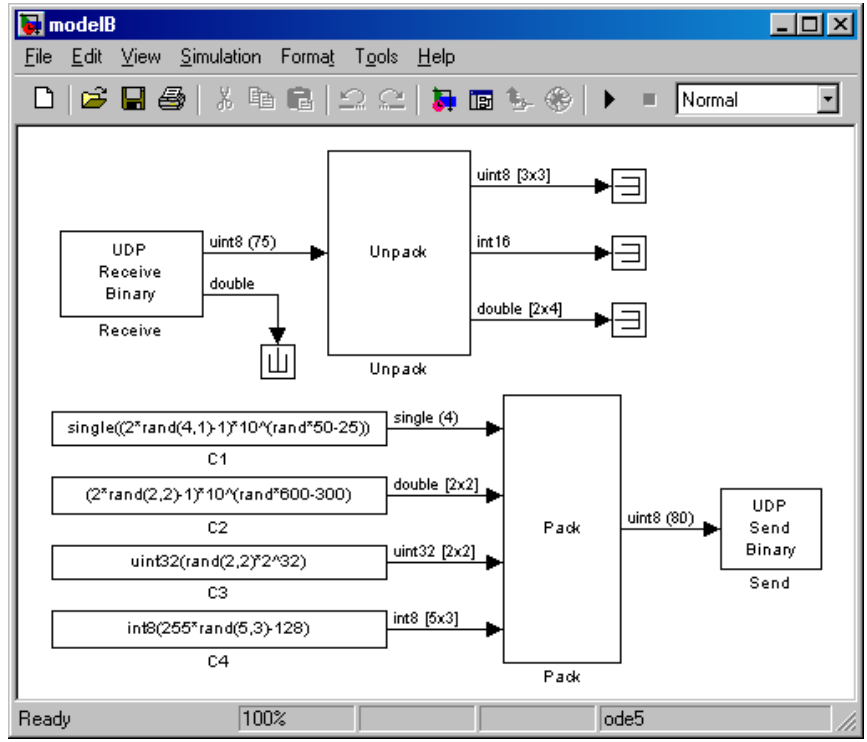
For setting up the Receive block, **IP address to receive from** is set to `192.168.0.1` (the hypothetical address of the system that will run Model B). The **IP port to receive from** is set to `25000` (the same value as set in the Send block in Model A). The **Output port width** is set to `75` which is obtained from the output port width of the Pack block in Model A.

For the Unpack block, **Byte Alignment** is set to `1`, and the **Output port datatypes** is set to `{'uint8','int16','double'}` from the Pack block in Model A. The **Output port dimensions** is set to `{[3 3],1,[2 4]}` from the dimensions of the inputs to the Pack block in modelA.

Note that in Model B, the second output port of the Receive block is sent into a terminator. This may be used to determine when a packet has arrived. The same is true for the outputs of the Unpack block, which in a real model would be used in the model.

For constructing the Model B to Model A side of the communication, follow an analogous procedure. The final models are shown below.





The following table lists the parameters in Model A.

Block	Parameter	Value
Receive	IP address	`192.168.0.2`
	IP port	`25000`
	Output port width	`80`
	Sampletime	`0.01`
Unpack	Output port dimensions	`{4,[2 2],[2 2],[5 3]}`

Block	Parameter	Value
	Output port datatypes	`{'single','double', 'uint32','int8'}`
	Byte Alingment	`2`

The following table lists the parameters in Model B

Block	Parameter	Value
Pack	Input Port Datatypes	`{'single','double', 'uint32','int8'}`
	Byte Alingment	`2`
Send	IP address	`192.168.0.1`
	IP port	`25000`
	Sampletime	`0.01`

Note on UDP Communication

The UDP blocks work in the background while the real-time application is not running. Also, the UDP communication has been set up to have a maximum of two UDP packets waiting to be read. All subsequent packets are rejected. This has been done to prevent excessive memory usage, and to minimize the load on the TCP/IP stack. Consequently, when any large background task is being performed, such as uploading a screen shot or communicating large pages through the WWW interface, packet loss may occur. Applications should be designed such that this is not critical. That is, the receipt of further packets after the ones that were lost will ensure graceful continuation.

4

Embedded Option

The xPC Target Embedde Option allows you to boot the target PC from an alternate device other than a floppy disk drive such as a hard disk drive or flash memory. It also allows you to create stand-alone applications on the target PC independent from the host PC.

This chapter includes the following sections:

Introduction

The xPC Target Embedded Option allows you to boot the xPC Target kernel from not only a floppy disk drive, but also from other devices, including a flash disk or a hard disk drive. By using xPC Target Embedded Option, you can configure target PCs to automatically start execution of your embedded application for continuous operation each time the system is booted. You use this capability to deploy your own real-time applications on target PC hardware.

The xPC Target Embedded Option extends the xPC Target base product by adding two additional modes of operation:

- **DOSLoader** — This mode of operation is used to start the kernel on the target PC from not only a floppy disk, but optionally start it from a flash disk or a hard disk. The target PC then waits for the host computer to download a real-time application either using the RS232 serial connection or using TCP/IP network communication. Control and setting of starting, stopping, parameters, tracing, and other properties can be achieved from either the host PC of from the target PC.
- **StandAlone** — This mode extends the DOSLoader mode. After starting the kernel on the target PC, StandAlone mode automatically starts execution of your target application for complete stand-alone operation. This eliminates the need for using a host computer and allows you to deploy real-time applications on PC hardware environments.

Whether you are using the xPC Target Embedded Option with the **DOSLoader** mode or the **StandAlone** mode, you initially boot your target PC with DOS from virtually any boot device. Then the kernel is invoked from DOS.

Note The xPC Target Embedded Option requires a boot device with DOS installed. DOS software and license are not included with xPC Target or with the xPC Target Embedded Option.

During setup of either the **DOSLoader** mode or **StandAlone** mode, the xPC Target Setup window allows you to create files for installation on the target boot device. One of these files is an `autoexec.bat` file. When DOS starts, it

invokes the `autoexec.bat` file which in turn starts the xPC Target kernel on the target PC.

If you do not provide an target application and an `autoexec.bat` file to invoke your target application, xPC Target Embedded Option starts the kernel on your target PC and is ready to receive your target application whenever you build and download a new one from the host computer.

In comparison, when using xPC Target without the xPC Target Embedded Option, you can only download real-time applications to the target PC after booting from an xPC Target boot disk. Because of this, when using xPC Target without Embedded Option is not available, you are always required to use a target PC equipped with a floppy disk drive. However, there are several cases where your target system may not have a floppy disk drive or where the drive is removed after setting up the target system. These cases can be overcome by using the DOSLoader mode.

DOSLoader Mode Overview

With the **DOSLoader** mode of operation, you first set up a boot device such as a floppy disk drive, flash disk, or a hard disk drive. This boot device must include DOS and modules from xPC Target Embedded Option. Once the kernel starts running, it awaits commands from the host computer and a target application that is downloaded from the host computer. The primary purpose of the **DOSLoader** mode is to allow you to boot from devices other than the floppy drive.

StandAlone Mode Overview

With the **StandAlone** mode of operation, you create completely stand-alone applications which start execution automatically when the target PC is booted. There is no need for communication with a host computer to downloaded the application after booting. Once the boot device has been set up with DOS, modules from xPC Target Embedded Option, and your target application, you boot the target PC. Upon booting, DOS invokes your `autoexec.bat` file which invokes the kernel. However, in **StandAlone** mode, your target application is combined with kernel in one binary `*.rtb` file. The final result is that your target application starts automatically each time the target PC is booted. By using xPC Target Embedded Option, you can deploy control systems, DSP applications, and other systems on PC hardware for use in production

applications using PC hardware. Typically these production applications are found in systems where production quantities are low to moderate.

xPC Target Embedded Option also gives you the choice of using target scopes on the target PC. When using **StandAlone** mode, target scopes allows you to trace signals using the target PC monitor without any interaction from the host computer. Assuming you do not want to view signals on the target PC, it is not necessary to use target scopes or a monitor on your target PC. In such a case, your system is able to operate as a *black-box* without a monitor, keyboard, or mouse. Stand-alone applications are automatically set to continue running for an infinite time duration or until the target computer is turned off.

Architecture

xPC Target Embedded Option creates additional files that you add to your target PC DOS boot device. With the **DOSLoader** mode, an `autoexec.bat` file is generated. This file enables DOS to automatically execute the file `xpcboot.com` when the target PC is booted. The file `autoexec.bat` includes an argument that invokes a `*.rtb` file containing the xPC Target kernel. Therefore, when the boot device invokes DOS, the `autoexec.bat` file then starts the xPC Target kernel. All of these files are placed on a floppy disk when you click **BootDisk** from the `xpcsetup` GUI. Your real-time application is not copied to the boot device. You create the real-time application later by clicking **Build**.

The **StandAlone** mode operates in a similar fashion with a few important differences. From the `xpcsetup` GUI, after choosing **StandAlone**, you only click **Update** to make your current selections active. When you later click **Build**, an `autoexec.bat` file and the `xpcboot.com` file are placed in a subdirectory that is created within your present working directory. This directory is named: *modename*`_xpc_emb`. In addition, the build process creates your target application and combines it with the xPC Target kernel. This combined `*.rtb` file is also placed in the same *modename*`_xpc_emb` subdirectory. You copy these files onto any DOS boot device. Then, upon booting DOS, the xpcboot.com file is invoked with the kernel and with your target application. If you choose to use target scopes with your stand-alone application, you can do so provided appropriate xPC Target Scope blocks are added and configured prior to code generation.

A small DOS executable called `xpcboot.com` is the core module of the Embedded Option. This module is used in both the **DOSLoader** mode and the

StandAlone mode. The module `xpcboot.com` is executed from DOS. It loads and executes any xPC Target application. The first argument given to `xpcboot.com` is the name of the image file (`*.rtb`) to be executed. This image file contains the xPC Target kernel and options, such as whether you are communicating using a serial cable or TCP/IP, and the ethernet address you have assigned to the target PC.

Since the xPC Target loader is just an ordinary xPC Target application, the loader can be executed from `xpcboot.com`.

Before starting the kernel, you must first boot the target PC under DOS. The module `xpcboot.com` is then automatically executed under DOS by `autoexec.bat`. To boot the target PC under DOS, you must first install DOS on the target PC boot device. The xPC Target Embedded Option does not have specific requirements as to the type of device you use to boot DOS. It is possible to boot from a floppy disk drive, hard disk drive, flash disk, or other device where you have installed DOS.

DOS is only needed to execute `xpcboot.com` and read the image file from the file system. After switching to the loaded kernel, and then executing the xPC Target application, DOS is discarded and is unavailable, unless you reboot the target PC without automatically invoking the xPC Target kernel. Once the xPC Target application begins execution, the target application is executed entirely in the protected mode using the 32-bit flat memory model.

Restrictions

The following restrictions apply to the booted DOS environment when you use `xpcboot.com` to execute the target applications:

- The CPU must be executed in real mode
- While loaded in memory, the DOS partition must not overlap the address range of a target application

You can satisfy these restrictions by avoiding the use of additional memory managers like `emm386` or `qemm`. Also, you should avoid any utilities that attempt to load in high memory space (for example, `himem.sys`). If the target PC DOS environment does not use a `config.sys` file or memory manager entries in the `autoexec.bat` file, there should not be any problems when running `xpcboot.com`.

It is also necessary that your **TargetMouse** setting is consistent with your hardware. Some PC hardware may use an RS232 port for the mouse, while others use a PS2 mouse. If a mouse is not required in your application, you may choose to select **None** as your setting for the **TargetMouse**.

Updating the xPC Target Environment

After the xPC Target Embedded Option software has been correctly installed, the xPC Target environment, visible through `xpcsetup` or `getxpcenv`, contains new property choices for **DOSLoader** or **StandAlone,** in addition to the default **BootDisk** that is normally used with xPC Target.

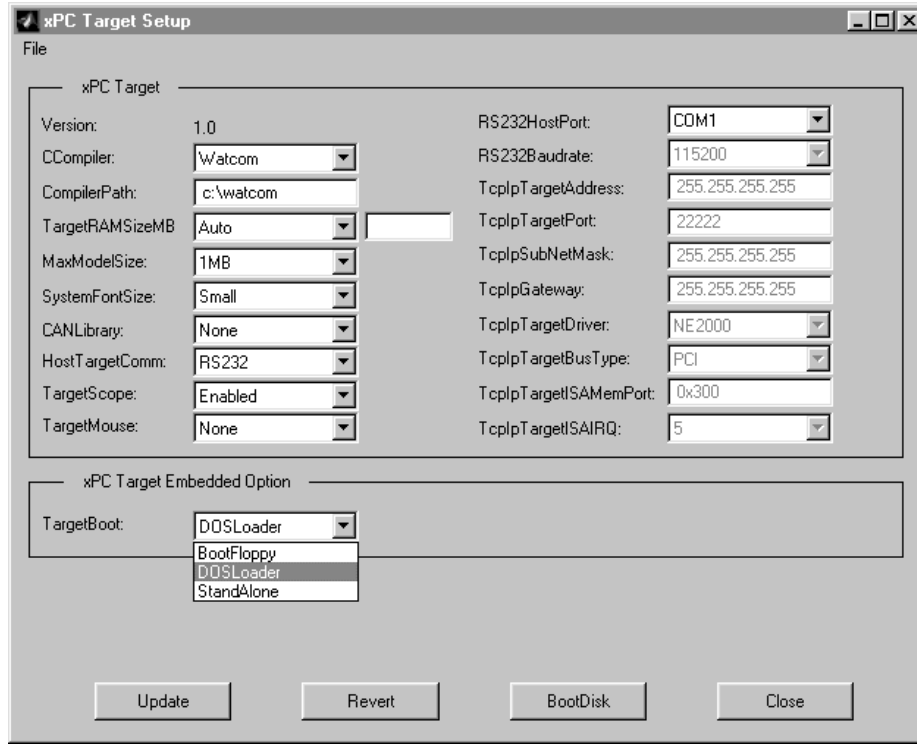
It is assumed that the xPC Target environment is already set up and working properly with the xPC Target Embedded Option disabled. If you have not already done so, we recommend you confirm this now.

You can use the function `getxpcenv` to see the current selection for **TargetBoot**, or you can view this through the xPC Target Setup window. Start MATLAB and execute the function

```
xpcsetup
```

Within the frame of the xPC Target Embedded Option, you see the property **TargetBoot,** as well as the currently selected value. The choices are:

- **BootFloppy** — Standard mode of operation when xPC Target Embedded Option is not installed.
- **DOSLoader** — For invoking the kernel on the target PC from DOS.
- **StandAlone** — For invoking the kernel on the target PC from DOS and automatically starting the target application without connecting to a host computer. With this mode, the kernel and the target application are combined as a single module that is placed on the boot device.



The default setting for the property **TargetBoot** is **BootFloppy**. When using **BootFloppy**, xPC Target must first create a target boot disk, which is then used to boot the target PC.

The property **TargetBoot** can be set to two other values, namely **DOSLoader** or **StandAlone**. If the xPC Target loader is booted from any boot device with DOS installed, the value **DOSLoader** must be set as shown above. If you want to use a stand-alone application that automatically starts execution of your target application immediately after booting, specify **StandAlone**.

After changing the property value, you need to update the xPC Target environment by clicking the **Update** button in the xPC Target Setup window. If your choice is **DOSLoader**, a new target boot disk must be created by clicking **BootDisk**. Note that this overwrites the data on the inserted target boot disk as new software modules are placed on the target boot disk. If your choice is **StandAlone**, you click the **Update** button. Upon building your next real-time application, all necessary xPC Target files are saved to a subdirectory below your present working directory. This subdirectory is named with your model name with the string "`_xpc_emb`" appended. For example, `xpcosc_xpc_emb`.

For more detailed information about how to use the xPC Target Setup, see Chapter 5, "Environment Reference."

Creating a DOS System Disk

When using DOSLoader mode, or StandAlone mode, you must first boot your target PC with DOS. These modes may be used from any boot device including flash disk, floppy disk drive, or a hard disk drive.

In order to boot DOS with a target boot disk, a minimal DOS system is required on the boot disk. With Windows 95, Windows 98, or DOS, you can create a DOS boot disk using the command

```
sys a:
```

Note xPC Target Embedded Option does not include a DOS license. You must obtain a valid DOS license for your target PC.

It is helpful to also copy additional DOS-utilities to the boot disk including

- A DOS-editor to edit files
- The format program to format a hard disk or FlashRAM
- The `fdisk` program to create partitions
- The `sys` program to transfer a DOS system onto another drive, such as the hard disk drive

A `config.sys` file is not necessary. The `autoexec.bat` file should be created to automatically boot the loader or a standalone xPC Target application. This is described in the following sections.

DOS Loader Target Applications

This section includes the following topics:

- "Creating a Target Boot Disk for DOS Loader"
- "Creating a Target Application for DOS Loader"

Creating a Target Boot Disk for DOS Loader

As the first step, we assume you have created a DOS system disk and updated the xPC Target environment by setting the property **TargetBoot** to **DOSLoader**. From the xPC Target Setup window, click the **BootDisk** button and xPC Target copies the necessary files to the DOS disk. The files that are added to the DOS boot disk include:

- checksum.dat
- autoexec.bat
- xpcboot.com
- `*.rtb` (where * is defined in the table below)

With the **DOSLoader** mode, the correct `*.rtb` file is added to the DOS disk according to the options specified in the following table.

Table 4-1: DOSLoader Mode *.rtb File Naming Convention

xPC Target environment	HostTargetComm: RS232	HostTargetComm: TCP/IP
TargetScope: Disabled	`xpcsto`*n*`.rtb`	`xpctto`*n*`.rtb`
TargetScope: Enabled	`xpcsgo`*n*`.rtb`	`xpctgo`*n*`.rtb`

Note The numeric value of *n* corresponds to the maximum model size. This value is either 1, 4, or 16 megabytes. The default value for *n* is 1, or a 1-megabyte maximum model size.

The file `autoexec.bat` is copied to the DOS disk. This file should contain at least the following line:

```
xpcboot xxx.rtb
```

where `xxx.rtb` is the file described in table 11-1. We recommend that you view this `autoexec.bat` file to confirm this.

Now the target boot disk can be removed from the host and put into the target PC disk drive. By rebooting the target PC, DOS is booted from the target boot disk and the `autoexec.bat` file with the result in the xPC Target loader being automatically executed. From this point onwards, the CPU runs in protected mode and DOS is discarded.

You can repeat this procedure as necessary. There are no restrictions on the number of xPC Target boot floppies that you can create. However, xPC Target and the xPC Target Embedded Option do not include DOS licenses. It is assumed that you will purchase valid DOS licenses for your target PCs from the supplier of your choice.

If the `xpcboot` command is not placed in the `autoexec.bat`, `xpcboot.com` is not executed when the target PC is booted. Instead, the target will be finished once it has booted DOS. You can then use the DOS environment to create a DOS partition on a hard disk, format it, and transfer `xpcboot.com` and `xxx.rtb` onto it. The `autoexec.bat` file can then be placed on the hard disk and edited so that it automatically boots the xPC Target loader the next time the target PC is booted. After this step the floppy disk drive can be removed from the system. The same procedure works with flash disks and other boot devices.

Creating a Target Application for DOS Loader

After having booted the target PC as described in the proceeding section, the target PC is ready to receive xPC Target applications from the host computer. Only now, these applications are received by the **DOSLoader** component of xPC Target. In every aspect, the **DOSLoader** mode will allow your target PC to operate just as it normally would when running the xPC Target after booting from a standard xPC Target boot disk. When you click **Build** for your model, the target application is downloaded to the target PC using the communication protocol as you specified earlier in the xPC Target Setup window.

Stand-Alone Target Applications

This section includes the following topics:

- "Creating a Target Application for Stand-Alone"
- "Creating a Target Boot Disk for Stand-Alone"
- "Using Target Scope Blocks with Stand-Alone"

Creating a Target Application for Stand-Alone

After selecting **StandAlone** as your **TargetBoot** entry, the xPC Target environment is ready to create completely stand-alone applications using the Real-Time Workshop **Build** button.

Once the build process has finished, a message is displayed confirming that a stand-alone application has been created. With the **StandAlone** mode, the download procedure is not automated. The files necessary for creating stand-alone operation are placed in a subdirectory below your working directory. You copy these files to your DOS boot device.

After the build process is complete, files in your subdirectory include:

- `model.rtb`. This image contains the xPC Target kernel and your target application.
- `autoexec.bat`. This file is automatically invoked by DOS. It then runs `xpcboot.com` and the `*.rtb` file.
- `xpcboot.com`. This file is a static file that is part of xPC Target Embedded Option.

Note We suggest setting the property **HostTargetComm** to **RS232** if property **TargetBoot** is set to **StandAlone**. This will use less memory than the TCP/IP setting. With either setting, **StandAlone** mode does not have any interaction with the host PC.

Creating a Target Boot Disk for Stand-Alone

After making a bootable DOS boot disk, the file `autoexec.bat` file must contain at least the following line

```
xpcboot model.rtb
```

where *model* is the name of your Simulink model.

These files should be copied to your DOS boot disk and inserted into the target drive. By rebooting the target PC, DOS is booted from the boot disk. The `autoexec.bat` file invokes the command string shown above which starts the kernel and the real-time application. Because of the stand-alone nature of the executed `rtb` file, the simulation of the xPC Target application starts immediately. Interaction between the host PC and target PC is no longer possible.

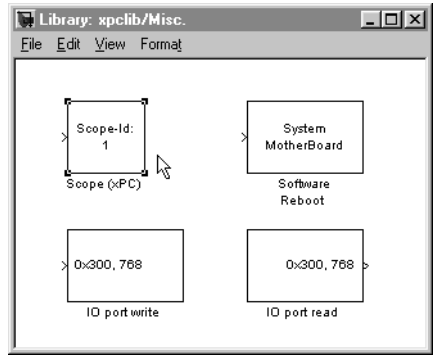
Is also possible to transfer the DOS system and stand-alone xPC Target applications to a hard disk or a flash RAM board. This offers great flexibility in creating self-starting stand-alone applications.

Using Target Scope Blocks with Stand-Alone

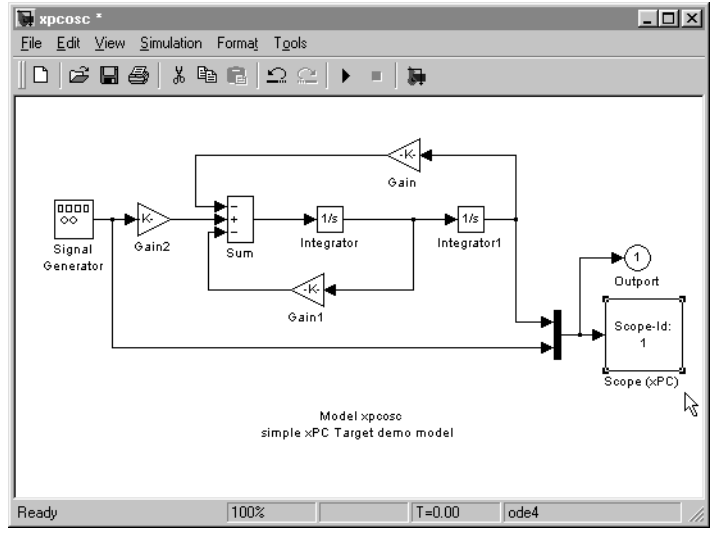
When using xPC Target Embedded Option with **StandAlone** mode, you can also use target scopes for tracing signals and displaying them on the target screen.

Because host-to-target communication is not supported with the **StandAlone** mode, scope objects of type `target` must be defined within the Simulink model before the xPC Target application is built.

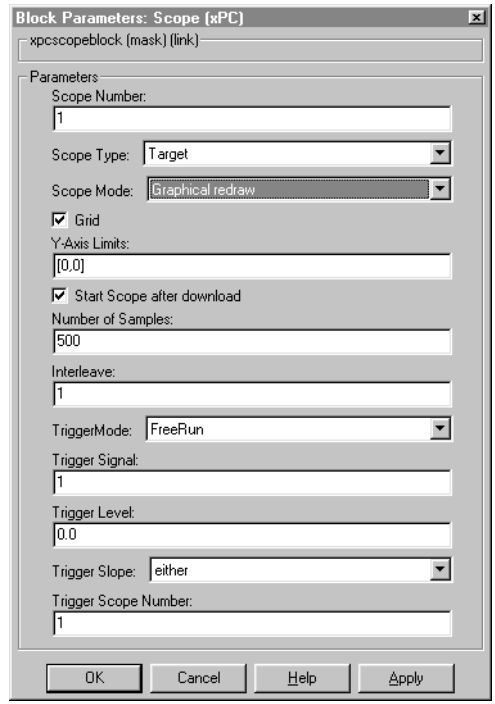
xPC Target (basic package) offers a block for such purposes.



Copy the **Scope (xPC)** block into your block diagram and connect the signals you would like to view to this block. Multiple signals can be used provided a **Mux** block is used to bundle them.



It is necessary to edit the **Scope (xPC)** dialog box and confirm that the check box entry for **Start Scope after download** is checked as shown in the following dialog box.



This setting is required to enable target scopes to begin operating as soon as the application starts running. The reason this setting is required is that the host PC is not available in StandAlone mode to issue a command that would start scopes. With these settings, click **Build** and copy the files from your *model name*_xpc_emb subdirectory to your boot disk. Then boot your target PC. When the target application starts to run, the target scopes will start automatically. A monitor is needed on your target PC to view the results.

Note When using target scopes with **StandAlone** mode, you must specify the Scope Type as Target prior to generating code.

5

Environment Reference

The xPC Target environment defines connections and communication between the host and target computers. It also, defines the build process for a real-time application.

This chapter includes the following sections:

- **"Environment"** — List of environment properties and functions with a brief description
- **"Using Environment Properties and Functions"** — Common tasks within the xPC Target software environment
- **"System Functions"** — List of functions for testing and opening graphical interfaces

Environment

The xPC Target environment defines the software and hardware environment of the host PC as well as the target PC. An understanding of the environment properties will help you to correctly configure the xPC Target environment.

This section includes the following topics:

- **"Environment Properties"** — List of properties with a brief description
- **"Environment Functions"** — List of functions with a brief description

Environment Properties

The environment properties define communication between the host PC and target PC, the type of C compiler and its location, and the type of target boot floppy created during the setup process. You can view and change these properties using the environment functions or Setup window.

Table 5-1: List of Environment Properties

Environment property	Description
`Version`	xPC Target version number. Read-only.
`Path`	xPC Target root directory. Read-only.
`CCompiler`	Values are `'Watcom'` or `'VisualC'`. From the Setup window **CCompiler** list, choose either **Watcom** or **VisualC**.
`CompilerPath`	Value is a valid compiler root directory. Enter the path where you installed a Watcom C/C++ or Microsoft Visual C/C++ compiler. If the path is invalid or the directory does not contain the compiler, then when you use the function `updatexpcenv` or build a target application, an error message appears.

Table 5-1: List of Environment Properties

Environment property	Description
`TargetRAMSizeMB`	Values are `'Auto'` or `'`*`MB of target RAM`*`'`. From the Setup window **TargetRAMSizeMB** list, choose either **Auto** or **Manual**. If you select **Manual**, enter the amount of RAM, in megabytes, installed on the target PC. This property is set by default to **Auto**. **TargetRAMSizeMB** defines the total amount of installed RAM in the target PC. This RAM is used for the, kernel, target application, data logging, and other functions that use the heap. If **TargetRAMSizeMB** is set to **Auto**, the target application automatically determines the amount of memory up to 64 MB. If the target PC does not contain more than 64 MB of RAM, or you do not want to use more then 64 MB, select **Auto**. If the target PC has more than 64 MB of RAM, and you want to use more than 64 MB, select **Manual**, and enter the amount of RAM installed in the target PC.
`MaxModelSize`	Values are `'1MB'`, `'4MB'`, or `'16MB'`. From the Setup window **MaxModelSize** list, choose either**1 MB**, **4 MB**, or **16 MB**. Choosing the maximum model size reserves the specified amount of memory on the target PC for the target application. The remaining memory is used by the kernel and by the heap for data logging. Selecting too high a value leaves less memory for data logging. Selecting too low a value does not reserve enough memory for the target application and creates an error.

Table 5-1: List of Environment Properties

Environment property	Description
`SystemFontSize`	Values are `'Small'` or `'Large'`. From the Setup window **SystemFontSize** list, choose either **Small**, or **Large**. The xPC Target GUIs use this information to change the font size.
`CANLibrary`	Values are `'None'`, `'200 ISA'`, `'527 ISA'`, `'1000 PCI'`, `'1000 MB PCI'`, or `'PC104'`. From the Setup window **CANLibrary** list, choose **None**, **200 ISA**, **527 ISA**, **1000 PCI**, **1000 MB PCI**, or **PC104**.
`HostTargetComm`	Values are `'RS232'` or `'TcpIp'`. From the Setup window **HostTargetComm** list, choose either **RS232** or **TCP/IP**. If you select **RS232**, you also need to set the property **RS232HostPort**. If you select **TCP/IP**, then you also need to set all properties that start with **TcpIp**.
`RS232HostPort`	Values are `'COM1'` or `'COM2'`. From the Setup window **RS232HostPort** list, choose either **COM1** or **COM2** for the connection on the host computer. xPC Target automatically determines the COM port on the target PC. Before you can choose an RS232 port, you need to set the **HostTargetComm** property to **RS232**.
`RS232Baudrate`	Values are `'115200'`, `'57600'`, `'38400'`, `'19200'`, `'9600'`, `'4800'`, `'2400'`, or `'1200'`. From the **RS232Baudrate** list, choose **115200**, **57600**, **38400**, **19200**, **9600**, **4800**, **2400**, or **1200**.

Table 5-1: List of Environment Properties

Environment property	Description
`TcpIpTargetAddress`	Value is '*xxx.xxx.xxx.xxx*'. In the Setup window **TcpIpTargetAddress** box, enter a valid IP address for your target PC. Ask you system administrator for this value. For example, `192.168.0.1`
`TcpIpTargetPort`	Value is '*xxxxx*'. In the Setup window **TcpIpTargetPort** box, enter a value greater than `20000`. This property is set by default to `22222` and should not cause any problems. The number is higher than the reserved area (telnet, ftp, ...) and it is only of use on the target PC.
`TcpIpSubNetMask`	Value is '*xxx.xxx.xxx.xxx*'. In the Setup window **TcpIpSubNetMask** text box, enter the subnet mask of your LAN. Ask you system administrator for this value. For example, `255.255.0.0`.

Table 5-1: List of Environment Properties

Environment property	Description
`TcpIpGateway`	Value is '*xxx.xxx.xxx.xxx*'. In the Setup window **TcpIpGateway** box, enter the IP address for your gateway. This property is set by default to `255.255.255.255` which means that a gateway is not used to connect to the target PC. If you communicate with your target PC from within a LAN that uses gateways, and your host and target computers are connected through a gateway, then you need to enter a value for this property. If your LAN does not use gateways, you do not need to change this property. Ask your system administrator.
`TcpIpTargetDirver`	Values are `'NE2000'` or `'SMC91C9X'`. From the Setup window **TcpIpTargetDriver** list, choose either **NE2000** or **SMC91C9X**. The Ethernet card provided with xPC Target uses the **NE2000** driver.
`TcpIpTargetBusType`	Values are `'PCI'` or `'ISA'`. From the Setup window **TcpIpTargetBusType** list, choose either **PCI** or **ISA**. This property is set by default to **PCI**, and determines the bus type of your target PC. You do not need to define a bus type for your host PC, which can be the same or different from the bus type in your target PC. If **TcpIpTargetBusType** is set to **PCI**, then the properties **TcpIpISAMemPort** and **TcpIpISAIRQ** have no effect on TCP/IP communication. If you are using an ISA bus card, set **TcpIpTargetBusType** to **ISA** and enter values for **TcpIpISAMemPort** and **TcpIpISAIRQ**.

Table 5-1: List of Environment Properties

Environment property	Description
`TcpIpTargetISAMemPort`	Value is *'0xnnnn'*. If you are using an ISA-bus Ethernet card, you must enter values for the properties **TcpIpISAMemPort** and **TcpIpISAIRQ**. The values of these properties must correspond to the jumper settings or ROM settings on your ISA-bus Ethernet card. On your ISA-bus card, assign an IRQ and I/O-port base address by moving the jumpers on the card. We recommend setting the I/O-port base address to around 0x300. If one of these hardware settings leads to a conflict in your target PC, choose another I/O-port base address and make the corresponding changes to your jumper settings.
`TcpIpTargetISAIRQ`	Value is *'n'* where *n* is between 4 and 15. If you are using an ISA bus Ethernet card, you must enter values for the properties **TcpIpISAMemPort** and **TcpIpISAIRQ**. The values of these properties must correspond to the jumper settings or ROM settings on the ISA-bus Ethernet card. On your ISA-bus card, assign an IRQ and I/O-port base address by moving the jumpers on the card. We recommend setting the IRQ to 5, 10, or 11. If one of these hardware settings leads to a conflict in your target PC, choose another IRQ and make the corresponding changes to your jumper settings.

Table 5-1: List of Environment Properties

Environment property	Description
`EmbeddedOption`	Values are `'Disabled'` or `'Enabled'`. This property is read-only. **Note** The xPC Target Embedded Option is enabled only if you purchase an additioanl license.
`TargetScope`	Values are `'Disabled'` or `'Enabled'`. From the Setup window **TargetScope** list, choose either **Enabled** or **Disabled**. The property **TargetScope** is set by default to **Enabled**. If you set **TargetScope to Disabled**, then the target PC displays information as text. To use all of the features of target scope, you also need to install a keyboard and mouse on the target PC.

Table 5-1: List of Environment Properties

Environment property	Description
`TargetMouse`	Values are `'None'`, `'PS2'`, `'RS232 COM1'`, `'RS232 COM2'`. From the Setup window **TargetMouse** list, choose **None**, **PS2**, **RS232 COM1**, or **RS232 COM2**. Before you can select a target mouse, you need to set the **Target Scope** property to **Enabled**. **TargetMouse** allows you to disable or enable mouse support on the target PC: • If you do not connect a mouse to the target PC, you need to set this property to **None**, otherwise, the target application may not behave properly. • If the target PC supports PS/2 devices (keyboard and mouse) and you connect a PS/2 mouse, set this property to **PS2**. • If you connect a serial RS232 mouse to the target PC, choose either **RS232 COM1** or **RS232 COM2** depending on which serial port you attached the mouse.
`TargetBoot`	Values are `'BootFloppy'`, `'DOSLoader'`, or `'StandAlone'`. From the Setup window **TargetBoot** list, choose either **BootFloppy**, **DOSLoader**, or **StandAlone**. If your license file does not include the license for the xPC Target Embedded Option, the **Target Boot** box is disabled with **BootFloppy** as your only selection. With the xPC Target Embedded Option licensed and installed, you have the additional choices of **DOSLoader** and **StandAlone**.

Environment Functions

The environment functions allow you to change the environment properties. The functions are listed in the following table.

Table 5-2: List of Environment functions

Environment functions	Description
`getxpcenv`	List environment properties in the MATLAB window or assign the list as a cell array to a MATLAB variable.
`setxpcenv`	First of two steps to change environment properties. See also updatexpcenv.
`updatexpcenv`	Changes the current environment properties to equal the new properties entered using the function setxpcenv.
`xpcbootdisk`	Creates a boot floppy disk containing the kernel according to the current environment properties.

Using Environment Properties and Functions

You use the xPC Target Setup window to enter properties that are independent of your model.

This section includes the following topics:

Changing Environment Properties

- "Getting a List of Environment Properties"
- "System Functions"
- "Changing Environment Properties with Graphical Interface"
- "Changing Environment Properties with Command Line Interface"

Target Boot Disk

- "Creating a Target Boot Disk with Graphical Interface"
- "Creating a Target Boot Disk with Command Line Interface"

To enter properties specific to your model and its build procedure, see "Entering the Simulation Parameters" on page 3-8. These properties are saved with your Simulink model.

Getting a List of Environment Properties

To use the xPC Target functions to change environment properties, you need to know the names and allowed values of the properties. Use the following procedure to get a list of the property names, their allowed values, and their current values:

1 In the MATLAB window, type

```
setxpcenv
```

MATLAB displays a list of xPC Target environment properties and the allowed values. For a list of the properties, see "Environment Properties" on page 5-3

2 Type

```
getxpcenv
```

MATLAB displays a list of xPC Target environment properties and the current values.

3 Alternately, in the MATLAB window, type

```
xpcsetup
```

MATLAB opens the xPC Target Setup window with the current values.

Saving and Loading the Environment

This feature makes it easy and fast to switch between different xPC Target environments:

1 In the xPC Target Setup window, and from the **File** menu, click **Save Settings**.

The Save xPC Target Environment dialog box opens.

2 Enter the name of an environment file (`*.mat`). Select a directory, and then click **Save**.

xPC Target saves the current environment properties.

After you have saved an xPC Target environment, you can load those property values back into xPC Target.

3 From the **File** menu, click **Load Settings**.

The Load xPC Target Environment dialog box opens.

4 Select a directory with a previously saved environment file (*.mat). Select the file, and then click **Open**.

5 In the xPC Target Setup window, click the **Close** button.

If you changed the environment properties, but do not click the Update button, xPC Target displays a warning.

Even if you decide to continue with the exit process, you will not lose the values you changed. However, the current environment does not reflect the changes you made in the xPC Target Setup window. If you reopen the xPC Target Setup window, the changes you made reappear, and the **Update** button is enabled.

Changing Environment Properties with Graphical Interface

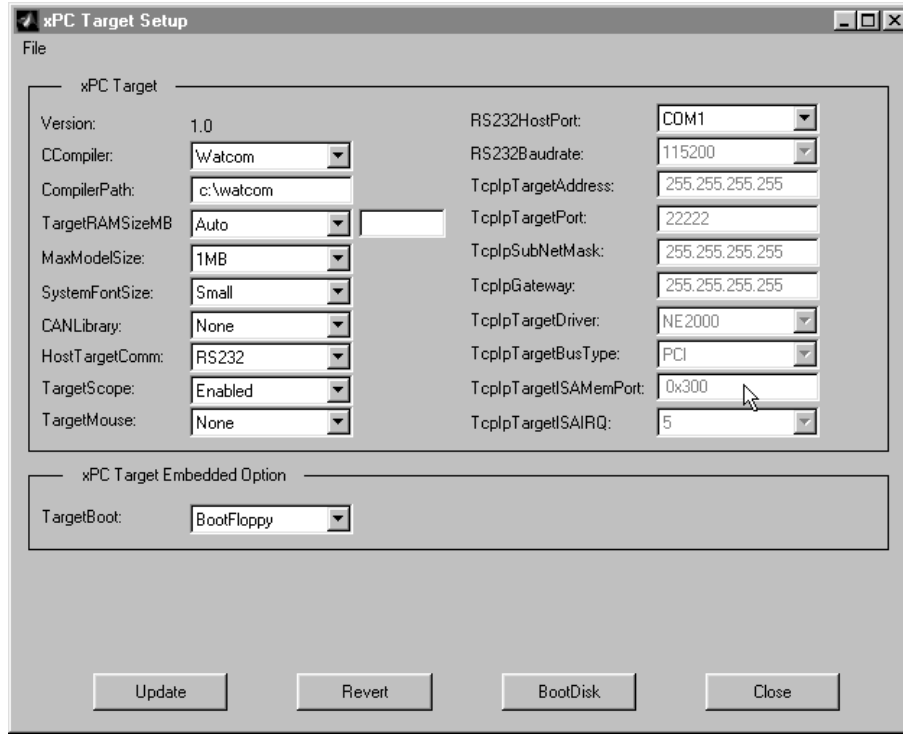
xPC Target lets you define and change environment properties. These properties include, the path to the C/C++ compiler, the host PC COM-port, the logging buffer size, and many others. Collectively these properties are known as the xPC Target environment.

To change an environment property using the xPC Target GUI, use the following procedure:

1 In the MATLAB command window, type

```
xpcsetup
```

MATLAB opens the xPC Target Setup window.



The xPC Target Setup window has two sections:

- **xPC Target**
- **xPC Target Embedded Option**

If your license does not include the xPC Target Embedded Option, the **TargetBoot** box is grayed-out with **BootFloppy** as your only selection. With the xPC Target Embedded Option, you have the additional choices of **DOSLoader** and **StandAlone**.

2 Change properties in the environment by entering new property values in the text boxes or choosing items from the lists.

After you make changes to the environment properties, you need to update the xPC Target environment. Updating makes your changes in the xPC Target Setup window equal to the current property values.

3 Click the **Update** button.

xPC Target updates the xPC Target environment and disables (grays-out) the **Update** button. As long as the **Update** button is enabled, the xPC Target environment needs to be updated.

Changing Environment Properties with Command Line Interface

xPC Target lets you define and change different properties. These properties include, the path to the C/C++ compiler, the host COM-port, the logging buffer size, and many others. Collectively these properties are known as the xPC Target environment.

You can use the command line functions to write an M-file script that accesses the environment settings according to your own needs. For example, you could write an M-file that switches between two targets.

The following procedure shows how to change the COM port property for your host PC from COM1 to COM2:

1 In the MATLAB window, type

```
setxpcenv('RS232HostPort','COM2')
```

The up-to-date column shows the values that you have changed, but have not updated.

```
HostTargetComm                  :RS232          up to date
RS232HostPort                   :COM2           COM2
RS232Baudrate                   :115200         up to date
```

Making changes using the function `setxpcenv` does not change the current values until you enter the update command.

2 In the MATLAB window, type

```
updatexpcenv
```

The environment properties you changed with the function `setxpcenv` become the current values.

```
HostTargetComm                  :RS232          up to date
RS232HostPort                   :COM2           up to date
RS232Baudrate                   :115200         up to date
```

Creating a Target Boot Disk with Graphical Interface

You use the target boot disk to load and run the xPC Target kernel.

After you make changes to the xPC Target environment properties, you need to create or update a target boot disk. To create a target boot disk for the current xPC Target environment, use the following procedure:

1 In the MATLAB window, type

```
xpcsetup
```

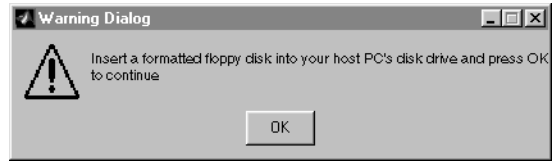
The xPC Target Setup window opens.

2 Click the **BootDisk** button.

If you didn't update the current settings, the following message box appears.

Click No. Click the Update button, and then click the BootDisk button again.

After you update the current properties, and click the **BootDisk** button, the following message box appears.



3 Insert a formatted floppy disk into the host PC disk drive, and then click **OK**. All data on the disk will be erased.

The write procedure starts and while creating the boot disk, the MATLAB command window displays the following status information. On Windows

NT systems, the status information is displayed only at the end of the write process.

```
xPC Target DiskWrite Utility Version 1.1,(c) 1998-2000 The
MathWorks, Inc.Read File:
C:\MATLAB\TOOLBOX\RTW\TARGETS\XPC\XPC\BIN\
..\..\target\kernel\xpcsgb1.rtd

Write Track  0- 9: ........uuuuuuuuuuuu
Write Track 10-19: uuuuuuuuuuuuuuuuuuuu
Write Track 20-29: uuuuuuuuuuuuuuuuuuuu
Write Track 30-39: uuuuuuuuuuuuuuuuuuuu
Write Track 40-49: uuuuuuuuuuuuuuuuuuuu
Write Track 50-59: uuuuuuuuuuuuuuuuuuuu
Write Track 60-69: uuuuuuuuuuuuuuuuuuuu
Write Track 70-79: uuuuuuuuuuuuuuuuuuuu
```

The process of creating a boot disk takes about 1 to 2 minutes.

4 Close the xPC Target Setup window.

5 Remove the target boot disk from the host PC disk drive. You may now use this disk to boot your target PC.

To create a boot disk using the command line interface, see “Creating a Target Boot Disk with Command Line Interface” on page 5-19.

Creating a Target Boot Disk with Command Line Interface

You use the target boot disk to load and run the xPC Target kernel.

After you make changes to the xPC Target environment properties, you need to create or update a boot disk. To create a target boot disk for the current xPC Target environment, use the following procedure:

1 In the MATLAB window, type

```
xpcbootdisk
```

xPC Target displays the following message.

```
Insert a formatted floppy disk into your host PC's
disk drive and press any key to continue.
```

2 Insert a formatted floppy disk into the host PC disk drive, and then press any key.

The write procedure starts and while creating the boot disk, the MATLAB command window displays the following status information. On Windows NT systems, the status information is displayed only at the end of the write process.

```
xPC Target Disk Write Utility Version 1.1,
(c) 1998-1999 The MathWorks Inc.
Read File: C:\MATLAB\TOOLBOX\RTW\TARGETS\XPC\XPC\BIN\..\..
\target\kernel\xpcsgb1.rtd

Write Track  0- 9: ..................
Write Track 10-19: ....uuuuuuuuuuuuuuuu
Write Track 20-29: uuuuuuuuuuuuuuuuuuuu
Write Track 30-39: uuuuuuuuuuuuuuuuuuuu
Write Track 40-49: uuuuuuuuuuuuuuuuuuuu
Write Track 50-59: uuuuuuuuuuuuuuuuuuuu
Write Track 60-69: uuuuuuuuuuuuuuuuuuuu
Write Track 70-79: uuuuuuuuuuuuuuuuuuuu
```

To create a boot disk using the graphical interface, see "Creating a Target Boot Disk with Graphical Interface" on page 5-17.

System Functions

The system functions allow you to open xPC Target GUIs and run tests from the MATLAB window.

This section includes the following topics:

- "GUI Functions"
- "Test Functions"
- "xPC Target Demos"

GUI Functions

The GUI functions are listed in the following table.

Table 5-3: List of GUI Functions

System functions	Description
`xpcscope`	Opens the scope manager window on the host PC for scopes with type host.
`xpcsetup`	Opens the Setup window.
`xpctargetspy`	Open the Target Spy window on the host PC. Use this GUI to upload the target PC screen to the host PC.
`xpctest`	Test the xPC Target installation.
`xpcscope`	Opens the scope manager window on the host PC for scopes with type host.
`xpcsetup`	Opens the Setup window.
`xpctargetspy`	Open the Target Spy window on the host PC. Use this GUI to upload the target PC screen to the host PC.
`xpctest`	Test the xPC Target installation.

Test Functions

The test functions are listed in the following table.

Table 5-4: List of Test Function

System functions	Description
`getxpcpci`	Determine which PCI boards are installed in the target PC.
`xpctargetping`	Test the communication between the host PC and the target PC
`xpctargetspy`	Open the Target Spy window on the host PC. Use this GUI to upload the target PC screen to the host PC.
`xpctest`	Test the xPC Target installation.

xPC Target Demos

The xPC Target demos are used to demonstrate the features of xPC Target. But they are also M-file scripts that you can view to understand how to write your own scripts for creating and testing target applications.

The following table lists the demo scripts that we provide with xPC Target.

Demo	Filename
Parameter Sweep	`parsweepdemo`
Signal tracing using free-run mode	`scfreerundemo`
Signal tracing using software triggering	`scsoftwaredemo`
Signal tracing using signal triggering	`scsignaldemo`
Signal tracing using scope triggering	`scscopedemo`
Signal tracing using the target scope.	`tgscopedemo`

To locate or edit a demo script

1 In the MATLAB window, type

```
scfreerundemo
```

MATLAB displays the location of the M-file.

```
D:\MATLAB\toolbox\rtw\targets\xpc\xpcdemos\scfreerundemo.m
```

2 Type

```
edit scfreerundemo
```

MATLAB opens the M-file in a MATLAB editing widow.

Environment and System Function Reference

This section includes an alphabetical listing of the environment and system functions.

For a list of the functions with a brief description, see:

- "Environment Functions" on page 5-11
- "GUI Functions" on page 5-20
- "Test Functions" on page 5-21

getxpcenv

Purpose

List environment properties assign to a MATLAB variable

Syntax

MATLAB Command Line

```
getxpcenv
```

Description

Function for environment properties. This function displays, in the MATLAB window, the property names, the current property values, and the new property values set for the xPC Target environment.

Examples

Return the xPC Target environment in the structure shown below. The output in the MATLAB window is suppressed. The structure contains three fields for property names, current property values, and new property values.

```
env =getxpcenv
```

```
env =
      propname: {1x25 cell}
    actpropval: {1x25 cell}
    newpropval: {1x25 cell}
```

Display a list of the environment property names, current values, and new values.

```
env =getxpcenv
```

See Also

The xPC Target functions `setxpcenv`, `updatexpcenv`, `xpcbootdisk`, and `xpcsetup`.

Purpose

Determine which PCI boards are installed in the target PC

Syntax

MATLAB Command Line

```
getxpcpci('type_of_boards')
```

Arguments

`type_of_boards`	Values are no arguments, `'all'` and `'supported'`.

Description

The information is displayed in the MATLAB window. Only devices supported by driver blocks in the xPC Target Block Library are displayed. The information includes the PCI bus number, slot number, assigned IRQ number, manufacturer name, board name, device type, manufacturer PCI Id, and the board PCI Id itself.

For a successful query:

- The host-target communication link must be working. (The function `xpctargetping` must return `success` before using the function `getxpcpci`.
- Either a target application is loaded or the loader is active. The latter is used to query for resources assigned to a specific PCI device, which have to be provided to a driver block dialog box prior to the model build process.

Examples

Return the result of the query in the struct `pcidevs` instead of displaying it. The struct `pcidevs` is an array with one element for each detected PCI device. Each element combines the information by a set of fieldnames. The struct contains more information compared to the displayed list, such as the assigned base addresses, the base and sub class.

```
pcidevs = getxpcpci
```

Display the supported and installed PCI devices.

```
getxpcpci('all')
```

Display the installed PCI devices, not only the devices supported by the xPC Target Block Library. This will include graphics controller, network cards, SCSI cards and even devices which are part of the motherboard chipset (for example PCI-to-PCI bridges).

```
getxpcpci('all')
```

getxpcpci

Display a list of the currently supported PCI devices in the xPC Target block library. The result is stored in a struct instead of displaying it.

```
getxpcpci('supported')
```

Purpose

Change xPC Target environment properties.

Syntax

MATLAB Command Line

```
setxpcenv('property_name', 'property_value')
setxpcenv('prop_name1', 'prop_val1', 'prop_name2', prop_val2')
setxpcenv
```

Arguments

`property_name`	Not case sensitive. Property names can be shortened as long as they can be differentiated from the other property names.
`property_value`	Character string. Type `setxpcenv` without arguments to get a listing of allowed values. Property values are not case sensitive.

Description

Function for environment properties. Enter new environment properties. If the new value is different from the current value, the property is marked as having a new values. Use the function `updatexpcenv` to change the current properties to the new properties.

The function `setxpcenv` works similarly to the function `set` of the MATLAB Handle Graphics system. The function `setxpcenv` must be called with an even number of arguments. The first argument of a pair is the property name, and the second argument is the new property value for this property.

Using the function `setxpcenv` without arguments returns a list of allowed property values in the MATLAB window.

Examples

List the current environment properties. For a description of properties and allowed values, see “Environment Properties” on page 5-3.

```
setxpcenv
```

Change the host PC, serial communication port, to COM2.

```
setxpcenv('HostCommPort','COM2')
```

See Also

The xPC Target functions `getxpcenv`, `updatexpcenv`, `xpcbootdisk`, and `xpcsetup`. The procedures “Changing Environment Properties with Graphical

Interface” on page 5-14 and “Changing Environment Properties with Command Line Interface” on page 5-16.

Purpose

Change current environment properties to equal new properties

Syntax

MATLAB Command Line

```
updatexpcenv
```

Description

Function for environment properties. This procedure includes creating communication M-files as well as patching the xPC Target kernel and system DLL's. Calling the function `updatexpcenv` is necessary after new properties are entered with the function `setxpcenv`, but before creating a target boot floppy with the function `xpcbootdisk`.

See Also

The xPC Target functions `setxpcenv`, `getxpcenv`, `updatexpcenv`, `xpcbootdisk`, and `xpcsetup`. The procedures "Changing Environment Properties with Graphical Interface" on page 5-14 and "Changing Environment Properties with Command Line Interface" on page 5-16.

xpcbootdisk

Purpose

Create xPC Target boot disk, and confirm the current environment properties

Syntax

MATLAB Command Line

```
xpcbootdisk
```

Description

Function for environment properties. This function creates a xPC target boot floppy for the current xPC Target environment which has been updated with the function `updatexpcenv`. Creating an xPC Target boot floppy consists of writing the correct bootable kernel image onto the disk. You are asked to insert an empty, formatted floppy disk into the floppy drive.

All existing files are erased by the function `xpcbootdisk`. If the inserted floppy disk already is an xPC Target boot disk for the current environment settings, this function exits without writing a new boot image to the floppy disk. At the end, a summary of the creation process is displayed.

If you update the environment, you need to update the target boot floppy for the new xPC environment with the function xpcbootdisk.

Examples

To create a boot floppy disk, in the MATLAB window, type

```
xpcbootdisk
```

See Also

The xPC Target functions `setxpcenv`, `getxpcenv`, `updatexpcenv`, `xpcbootdisk`, and `xpcsetup`. See also, the procedures “Creating a Target Boot Disk with Graphical Interface” on page 5-17 and “Creating a Target Boot Disk with Command Line Interface” on page 5-19.

Purpose

Open a scope manager window on the host PC.

Syntax

MATLAB Command Line

```
xpcscope
```

Description

This graphical user interface (GUI) allows you to define scopes that display on your host PC, choose signals, and control the data acquisition process.

See Also

The xPC Target function `xpctgscope` and the procedures “Signal Tracing with xPC Target GUI” on page 3-26 and “Signal Tracing with xPC Target GUI (Target Manager)” on page 3-31.

xpcsetup

Purpose

Open the Setup window

Syntax

MATLAB Command Line

```
xpcsetup
```

Description

This graphical user interface (GUI) allows you to:

- Enter and change environment properties
- Create an xPC Target boot floppy disk

See Also

See also the functions setxpcenv, getxpcenv, updatexpcenv, xpcbootdisk, and the procedures “I/O boards — If you use I/O boards on the target PC, you need to correctly install the boards. See the manufactures literature for installation instructions.” on page 2-12, “Environment Properties for Network Communication” on page 2-20, and.

Purpose

Test communication between the host and target computers

Syntax

MATLAB Command Line

```
xpctargetping
```

Examples

Check for communication between the host PC and target PC.

```
xpctargetping
```

Description

Ping's the target PC from the host PC and returns either 'success' or 'failed'. If the xPC Target kernel is loaded, running, and communication is working properly, this function returns the value 'success'.

This function works with both RS232 and TCP/IP communication.

```
ans =
success
```

See Also

The xPC Target procedure “Testing and Troubleshooting the Installation” on page 2-26.

xpctargetspy

Purpose

Open an xPC Target Spy window on the host PC

Syntax

MATLAB Command Line

```
xpctargetspy
```

Description

This graphical user interface (GUI) allows you to upload displayed data from the target PC.

The behavior of this function depends on the value for the environment property TargetScope.

- If TargetScope is enabled, a single graphics screen is uploaded. The screen is not continually updated because of a higher data volume when a target graphics card is in VGA mode.

  To update the host screen with another target screen, move the pointer into the Spy window and left-click.

- If TargetScope is disabled, text output is continuously transferred every second to the host and displayed in the window.

Examples

To open the Target Spy window, in the MATLAB window, type

```
xpctargetspy
```

See Also

The xPC Target procedures “I/O boards — If you use I/O boards on the target PC, you need to correctly install the boards. See the manufactures literature for installation instructions.” on page 2-12 and “Environment Properties for Network Communication” on page 2-20.

Purpose

Test the xPC Target installation

Syntax

MATLAB Command Line

```
xpctest
xpctest('reboot_flag')
```

Arguments

`reboot_flag`	`noreboot`. Skips the reboot test. User this option if the target hardware does not support software rebooting. Value is 'noreboot'

Description

Series of xPC Target tests to check the correct functioning of the following xPC Target tasks:

- Initiate communication between the host and target computers.
- Reboot the target PC to reset the target environment.
- Build a target application on the host PC.
- Download a target application to the target PC.
- Check communication between the host and target computers using commands.
- Execute a target application.
- Comparing the results of a simulation and the target application run.

xpctest('noreboot') skips test 2. Use this option if target hardware does not support software rebooting.

Examples

If the target hardware does not support software rebooting, and to skip test 2, in the MATLAB window, type

```
xpctest('noreboot')
```

See Also

The procedures “Testing and Troubleshooting the Installation” on page 2-26 and “Test 1, Ping Target System Standard Ping” on page 2-27.

xpctgscope

Purpose

Open the target scope manager window.

Syntax

MATLAB Command Line

```
xpctgscope
```

Description

This graphical user interface (GUI) allows you to define scopes that display on your target PC, choose signals, and control the data acquisition process.

See Also

The xPC Target function xpcscope and the procedures “Signal Tracing with xPC Target GUI (Target Manager)” on page 3-31 and “Signal Tracing with xPC Target GUI” on page 3-26.

Purpose

Disconnect the target PC from the current client application

Syntax

MATLAB Command Line

```
xpcwwwenable
```

Description

Use this function to disconnect the target application from MATLAB before you connect to the Web browser. Also, you can use this function to connect to MATLAB after using a Web browser, or switch to another Web browsers.

6

Target Object Reference

Use target objects to run and control real-time applications on the target PC.

This chapter includes the following sections:

- **"Target Object"** — Definition, properties, and methods
- **"Using Target Objects"** — Changing properties, and running methods

Target Object

xPC Target uses a target object to represent the target application and target kernel. An understanding of the target object properties and methods will help you to control and test your application on the target PC.

This section includes the following topics:

- "What is a Target Object?"
- "Target Object Properties"
- "Target Object Methods"

What is a Target Object?

A target object on the host PC represents the interface to a target application and the kernel on the target PC. You use target objects to run and control the target application.

When you change a target object property on the host PC, information is exchanged with the target PC and the target application.

To create a target object:

- Build a target application. xPC Target creates a target object during the build process.
- Use the target object constructor function `xpc`. In the MATLAB window, type `tg = xpc`.

A target object has associated properties and methods specific to that object.

Target Object Properties

Target object properties let you access information from your target application and control its execution. You can view and change these properties using target object methods.

The properties for a target object are listed in the following table. This table includes a description of the properties and which properties you can change directly by assigning a value.

Table 6-1: List of Target Object Properties

Property	Description	Write
`Connected`	Communication status between the host PC and the target PC. Values are 'Yes' or 'No'.	
`Application`	Name of the Simulink model and target application build from that model.	
`Mode`	Type of Real-Time Workshop code generation. Values are `'Real-Time Singletasking'`, `'Real-Time Multitasking'`, or `'Accelerate'`. The default value is `'Real-Time Singletasking'`.	
	Note Even if you select Real-Time Multitasking, the actual mode can be Real-Time Singletasking. This happens if your model contains only one or two tasks and the sample rates are equal.	
`Status`	Execution status of your target application. Values are `'stopped'` or `'running'`.	
`CPUoverload`	CPU status for overload. If the target application requires more CPU time than the sample time of the model, this value is set from `'none'` to `'detected'` and the current run is stopped. Correcting CPUoverload requires either a faster processor or a larger sample time.	

Table 6-1: List of Target Object Properties

Property	Description	Write
`ExecTime`	Execution Time. Time, seconds, since your target application started running. When the target application stops, the total execution time is displayed.	
`SessionTime`	Time since the kernel started running on your target PC. This is also the elapsed time since you booted the target PC. Values are in seconds.	
`StopTime`	Time when the target application stops running. Values are in seconds. The original value is set in the Simulink Simulation Parameters dialog box. When the `ExecutionTime` reaches the `StopTime`, the application stops running.	Yes
`SampleTime`	Time between samples. This value equals the step size, in seconds, for updating the model equations and post the outputs.	Yes
`AvgTET`	Average task execution time. This value is an average of the measured CPU times, in seconds, to run the model equations and post outputs during each sample interval. Task execution time is nearly constant with minor deviations due to cache, memory access, interrupt latency, and multirate model execution.	
`MinTET`	Minimum task execution time. Corresponds to the fastest time (smallest time measured), in seconds, to update model equations and post outputs.	

Table 6-1: List of Target Object Properties

Property	Description	Write
`MaxTET`	Maximum task execution time. Corresponds to the slowest time (longest time measured), in seconds, to update model equations and post outputs.	
`ViewMode`	Displays either all scopes or a single scope on the target PC. Values are `'all'` or a single scope index. This property is active only if the environemnt property TargetScope is set to `enabled`.	Yes
`TimeLog`	Storage in the MATLAB workspace for the time or t-vector logged during execution of the target application.	
`StateLog`	Storage in the MATLAB workspace for the state or x-vector logged during execution of the target application.	
`OutputLog`	Storage in the MATLAB workspace for the output, or y-vector logged during execution of the target application.	
`TETLog`	Storage in the MATLAB workspace for a vector containing task execution times during execution of the target application. To enable logging of the TET, you need to check the Log Task Execution Time box located at Simulation Parameters dialog box> Real-Time Workshop page > Category:xPC Target code generation options group	

Table 6-1: List of Target Object Properties

Property	Description	Write
`MaxLogSamples`	Maximum number of samples for each logged signal within the circular buffers for `TimeLog`, `StateLog`, `OutputLog`, and `TETLog`. `StateLog` and `OutputLog` can have one or more signals. This value is calculated by dividing the **Signal Logging Buffer Size** by the number of logged signals. The **Signal Logging Buffer Size** box is located at Simulation Parameters dialog box> Real-Time Workshop page > Category:xPC Target code generation options group.	
`NumLogWraps`	The number of times the circular buffer wrapped. The buffer wraps each time the number of samples exceeds `MaxLogSamples`.	
`LogMode`	Controls which data points are logged. • Equi-distant time. Logs a data point at every time interval. Set value to `'normal'`. • Equi-distant amplitude. Logs a data point only when one of the output values from the `OutputLog` changes by a specified amplitude. Set value to a signal value.	Yes
`Scopes`	List of index numbers with one index for each scope.	
`NumSignals`	The number of signals from your Simulink model that are available to be viewed with a scope.	
`ShowSignals`	Flag set to view or hide the list of signals from your Simulink blocks. This list is shown when you display the properties for a target object. Values are `'on'` or `'off'`.	Yes

Table 6-1: List of Target Object Properties

Property	Description	Write
`Signals`	List of viewable signals. This list is visible only when ShowSignals is set to `'on'`. • Property name. S0, S1. . . • Property value. Value of the signal • Block Name. Name of the Simulink block the signal is from.	
`S#`	Property name for a signal.	
`NumParameters`	The number of parameters from your Simulink model that you can tune or change.	
`ShowParameters`	Flag set to view or hide the list of parameters from your Simulink blocks. This list is shown when you display the properties for a target object. Values are `'on'` or `'off'`.	Yes
`Parameters`	List of tunable parameters. This list is visible only when `ShowParameters` is set to `'on'`. • Property name. P0, P1. . . • Property value. Value of the parameter in a Simulink block. • Type. Datatype of the parameter. Always double. • Size. Size of the the parameter. For example, scalar, 1x2 vector, or 2x3 matrix. • Parameter name. Name of a parameter in a Simulink block. • Block name. Name of a Simulink block	Yes
`P#`	Property name for block parameter.	

Target Object Methods

The target object methods allow you to control a target application on the target PC from the host PC.

Target object methods are entered in the MATLAB window on the host PC. You can also control the target application from the target PC using target PC commands. See “Target PC Commands” on page 6-11.

The methods are listed in the following table.

Table 6-2: List of Target Object Methods

Method	Description
`xpc`	Creates a target object on the host PC (constructor).
`set`	Sets writable target object properties to the specified value.
`get`	Returns the value of readable properties from a target object.
`start`	Starts the execution of a target application on the target PC.
`stop`	Stops the execution of a target application on the target PC.
`load`	Downloads a target application from the host PC to the target PC.
`unload`	Unloads a target application from the target PC. If a target application is running, it is stopped and unloaded.
`addscope`	Creates a new scope with type `'host'` or `'target'` on the target PC.
`getscope`	Returns the properties of a previously created scope from the target PC. The scope properties can be assigned to a MATLAB variable to create a scope object.

Table 6-2: List of Target Object Methods

Method	Description
`remscope`	Removes a scope from the target PC. This method does not remove the scope object, on the host PC, that represent the scope.
`getparamid`	Returns the property name or index of a parameter from the target object.
`getsignalid`	Returns the property name or index of a signal from the target object.
`getlog`	Uploads and returns one of the data logs from the target PC to the host PC. TimeLog, StateLog, OutputLog, TETLog
`reboot`	Reboot the target PC. If a target application is running, the target application is stopped, and then the target PC is rebooted.
`close`	Close the serial connection to the target PC so that the host PC can use the COMM port for another device.

Target PC Commands

The target PC commands allow you to control a target application on the target PC from the target PC.

Target PC commands are entered in the target PC command window on the target PC. You can also control the target application from the host PC using target object methods. See “Target Object Methods” on page 6-9.

The commands are listed in the following table.

Table 6-3: List of Target PC Commands

Command	Description
`delallvar`	Delete all variables. Syntax: `delvar`
`delvar`	Delete a variable. Syntax: `delvar` *`variable_name`*
`getpar`	Displays the value of a block parameter using the parameter index. Syntax: `setpar` *`parameter_index`*
`getvar`	Display the value of a variable. Syntax: `getvar` *`variable_name`*
`P#`	Display or change the value of a block parameter. For example, `P2` or `P2=1.23e-4` Syntax: *`parameter_name`*, or *`parameter_name`* `=` *`floating_point_number`*
`S#`	Displays the value of a signal. For example, S2. Syntax: *`signal_name`*.
`sampletime`	Enter a new sample time. Syntax: `sampletime =` *`floating_point_number`*

Table 6-3: List of Target PC Commands

Command	Description
`setpar`	Changes the value of a block parameter using the parameter index. Syntax: `setpar` *`parameter_index`* `=` *`floating_point_number`*
`setvar`	Sets a variable to a value. Later you can use that variable to do a macro expansion. Syntax: `setvar` *`variable_name`* `=` *`target_pc_command`* For example, you can type setvar aa=startscope 2, setvar bb=stopscope 2
`showvar`	Display a list of variables. Syntax: `showvar`
`stoptime`	Enter a new stop time. Use `inf` to run the target application until you manually stop it or reset the target PC. Syntax: `stoptime =` *`floating_point_number`*
`viewmode`	Zoom in to one scope, or zoom out to all scopes. Syntax: `viewmode` *`scope_number`*, or `viewmode 'all'`

Using Target Objects

xPC Target uses a target object to represent the target application and target kernel. This section shows some of the common tasks that you use with target objects.

This section includes the following topics:

- "Displaying Target Object Properties"
- "Setting the Value of a Target Object Property from the Host PC"
- "Setting the Value of a Target Object Property from the Target PC"
- "Getting the Value of a Target Object Property"
- "Using the Method Syntax with Target Objects"

Displaying Target Object Properties

You may want to list the target object properties to monitor a target application. The properties include the execution time, and average task execution time.

After you build a target application and target object from a Simulink model, you can list the target object properties. This procedure uses the default target object name `tg` as an example:

1 In the MATLAB window, type

```
tg
```

The current target application properties are uploaded to the host PC, and MATLAB displays a list of the target object properties with the updated values.

Note the target objects properties for TimeLog, StateLog, OutputLog, and TETLog are not updated at this time.

For a list of target object properties with a description, see "Target Object Properties" on page 6-4.

Setting the Value of a Target Object Property from the Host PC

You can change a target object property by using xPC Target methods on the host PC.

With xPC Target you can use either a function syntax or an object property syntax. The syntax set(*target_object*, *property_name*, *new_property_value*) can be replaced by:

target_object.*propety_name* = *new_property_value*.

For example, to change the stop time mode for the target object `tg`:

1. In the MATLAB window, type

   ```
   tg.stoptime = 1000
   ```

2. Alternately, you could type

   ```
   set(tg, 'stoptime', 1000)
   ```

Parameters are also target object properties. For example, to change the frequency of the signal generator in the model xpcosc:

1. In the MATLAB window, type

   ```
   tg.p2 = 30
   ```

2. Alternately, you could type

   ```
   set(tg, 'p2', 30)
   ```

When you change a target object property, the new property value is downloaded to the target PC. The xPC Target kernel then receives the information and changes the behavior of the target application.

To get a list of the writable properties, type `set(target_object)`. The build process assigns the default name of the target object to `tg`.

Note Method names are case-sensitive and need to be complete, but property names are not case-sensitive and need not be complete as long as they are unique.

Setting the Value of a Target Object Property from the Target PC

You can type commands directly from a keyboard on the target PC. These commands create a temporary difference between the behavior of the target application and the properties of the target object. The next time you access the target object, the properties are updated from the target PC:

1. On the target PC keyboard, press C, or point the target mouse in the command window.

   The target PC activates the command window.

2. Type a target command. For example, to change the frequency of the signal generator (parameter 2) in the model `xpcosc`, type

   ```
   setpar 2=30
   ```

3. Change the stop time. For example to set the stop time to 1000 type

   ```
   stoptime = 1000
   ```

   The parameter changes are make to the target application but not to the target object. When you type any xPC Target command in the MATLAB command window, the target PC returns the present properties to the target object.

   Note The target PC command `setpar` does not work for vector parameters.

Getting the Value of a Target Object Property

You can list a property value in the MATLAB window, or assign that value to a MATLAB variable. With xPC Target you can use either a function syntax or an object property syntax.

The syntax get(*target_object*, *property_name*) can be replaced by

target_object.propety_name

For example, to access the start time:

1 In the MATLAB window, type

endrun = tg.stoptime

2 Alternately, you could type

endrun = get(tg,'stoptime') or tg.get('stoptime')

Signals are also target object properties. For example, to get the value of the Integrator1 signal from the model xpcosc:

1 In the MATLAB window, type

outputvalue= tg.S0

2 Alternately, you could type

outputvalue = get(tg, 's2') or tg.get('s2')

To get a list of readable properties, type *target_object*. Without assignment to a variable, the property values are listed in the MATLAB window.

Note Method names are case-sensitive and need to complete, but property names are not case-sensitive and need not be complete as long as they are unique.

Using the Method Syntax with Target Objects

Use the method syntax to run a target object method. The syntax *method_name*(*target_object*, *argument_list*) can be replaced with:

```
target_object.method_name(argument_list)
```

Unlike properties, for which partial but unambiguous names are permitted, method names must be entered in full, and in lowercase. For example, to add a scope of type target with a scope index of 1:

1. In the MATLAB window, type

   ```
   tg.addscope('target',1)
   ```

2. Alternately, you could type

   ```
   addscope(tg, 'target', 1)
   ```

addscope

Purpose

Creates one or more scopes on the target PC

Syntax

MATLAB command line

Creating a scope and scope object without assigning to a MATLAB variable.

```
addscope(target_object, 'scope_type', new_scope_index)
target_object.addscope('scope_type', new_scope_index).
```

Creating a scope, scope object, and assign to a MATLAB variable.

```
scope_object = addscope(target_object,'scope_type',
new_scope_index)
```

```
scope_object = target_object.addscope('target', new_scope_index)
```

Target PC command line - When using this command on the target PC, it is limited to adding a scope of type target.

```
addscope
addscope new_scope_index
```

Arguments

`target_object`	Name of a target object.
`scope_type`	Values are `'host'` or `'target'`. This argument is optional with `host` as the default value.
`new_scope_index`	Vector of new scope indices. This argument is optional with the next available integer in the target object property `Scopes` as the default value. If you enter a scope index for an existing scope object, the result is an error.

Description

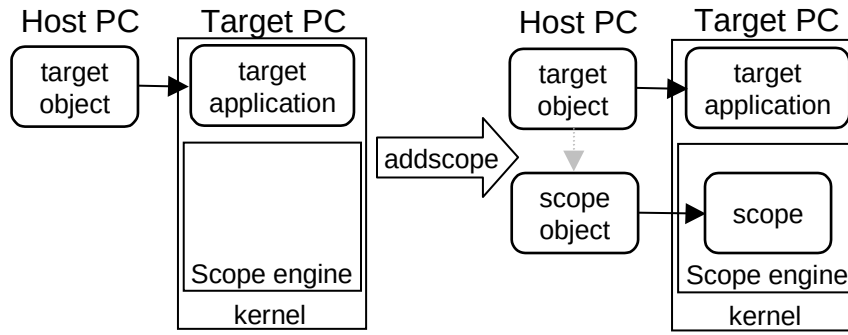
Method of a target object. Creates a scope on the target PC, a scope object on the host PC, and updates the target object property `Scopes`. This method returns a scope object vector. If the result is not assigned to a variable, the scope object properties are listed in the MATLAB window. If you try to add a scope with the same index as an existing scope, the result is an error.

A scope acquires data from the target application and displays that data on the target PC or uploads the data to the host PC.

All Scopes of type target or host run on the target PC

Scope of type target - Data collected is displayed on the target screen and acquisition of the next data package is initiated by the kernel.

Scope of type host - Collects data and waits for a command from the host PC for uploading the data. The data is then displayed using the host scope GUI (xpcscope) or other MATLAB functions.



Examples

Create a scope and scope object `sc1` using the method `addscope`. A target scope is created on the target PC with an index of 1, a scope object is created on the host PC, and it is assigned to the variable `sc1`. The target object property Scopes is changed from `No scopes defined` to `1`.

```
sc1 = addscope(tg,'target',1) or sc1 = tg.addscope('target',1)
```

Create a scope with the method `addscope` and then to create a scope object, corresponding to this scope, using the method `getscope`. A target scope is created on the target PC with an index of 1, a scope object is created on the host PC, but it is not assigned to a variable. The target object property Scopes is changed from `No scopes defined` to `1`.

```
addscope(tg,'target',1) or tg.addscope('target',1)
sc1 = getscope(tg,1) or sc1 = tg.getscope(1)
```

Create two scopes using a vector of scope objects `scvector`. Two target scopes are created on the target PC with a scope index of 1 and 2, two scope objects are created on the host PC that represent the scopes on the target PC. The target object property Scopes is changed from `No scopes defined` to `1,2`.

```
scvector = addscope(tg, 'target', [1, 2])
```

See Also

The xPC Target target object methods `remscope`, `getscope`. The xPC Target GUI function `xpcscope`. The xPC target M-file demo scripts listed in “xPC Target Demos” on page 5-21.

Purpose

Closes the serial port connecting the host PC with the target PC

Syntax

MATLAB command line

```
close(target_object)
```

Arguments

`target_object`	Name of a target object.

Description

Method of a target object. If you want to use the serial port for another function without quiting MATLAB, for example a modem, use this function to close the connection.

get

Purpose

Return the property values for target and scope objects.

Syntax

MATLAB command line

```
get(target_object, 'target_object_property')
```

Arguments

`target_object`	Name of a target object
`target_object_property`	Name of a target object property.

Description

Method of target objects. Gets the value of readable target object properties from a target object.

Examples

List the value for the target object property `StopTime`. Notice the property name is a string, in quotes, and not case-sensitive.

```
get(tg,'stoptime') or tg.get('stoptime')
```

```
ans = 0.2
```

See Also

The xPC Target target object method `set`. The scope object methods `get` and `set`. The built in MATLAB functions `get` and `set`.

Purpose

Get all or part of the output logs from the target object

Syntax

MATLAB command line

```
log = getlog(target_object, 'log_name', start_time,
number_points, interleave)
```

Arguments

log	User defined MATLAB variable.
`log_name`	Values are `TimeLog`, `StateLog`, `OutputLog`, or `TETLog`. This argument is required.
first_point	First data point. The logs begin with 1. This argument is optional. Default is 1
`number_points`	Number of points after the start time. This argument is optional. Default is all points in log.
`interleave`	1 returns all sample points. n returns every nth sample point. This argument is optional, Default is 1.

Description

Method of a target object. Use this function instead of the function `get` when you want only part of the data.

Examples

`To get the first 1000 points in a log.`

```
Outlog = getlog(tg, 'TETLog', 0, 1000)
```

To get every other point in the output log and plot values.

```
Output_log = getlog(tg, TETLog, 0, ,2)
Time_log = getlog(tg, TimeLog, 0, ,2)
plot(time_log, output_log)
```

See Also

The xPC Target target object methods `get`. The procedures "Entering the Simulation Parameters" on page 3-8, "Entering the Simulation Parameters" on page 3-8.

getparamid

Purpose

Get a parameter index or property name from the parameter list

Syntax

MATLAB command line

```
getparamid(target_object, 'block_name', 'parameter_name)
getparamid(target_object, 'block_name', 'parameter_name',
'flag')
```

Arguments

target_object	Name of a target object. The default name is tg.
block_name	Simulink block path and name.
parameter_name	Name of a parameter within a Simulink block
flag	If flag = property, then return the property name for the parameter. If flag = numeric, then return a number index. This argument is optional with the default behavior to return a property name.

Description

Method of a target object. Returns the index of a parameter in the parameter list based on the path to the parameter name. The names must be entered in full and are case-sensitive.

Examples

Get the property name for the parameter Gain in the Simulink block Gain1, incrementally increase gain, and pause to observe signal trace.

```
id = getparamid(tg, 'Subsyste/Gain1', 'Gain')
for i = 1 : 3
  set(tg, id, i*2000);
  pause(1);
end
```

Get the property name (P0, P1, . . .) of a single block.

```
getparamid(tg, 'Gain1', 'Gain')
```

Get the property index (0, 1, . . .) of asingal block.

```
getparamid(tg, 'Gain1', 'Gain', 'numeric')
```

P5 is a property of the target object. For example, you could assign a value to the gain with tg.p5 = 1000.

See Also

The xPC Target scope object method `getsignalid`. The xPC target M-file demo scripts listed in “xPC Target Demos” on page 5-21.

getscope

Purpose

Gets a scope object pointing to a scope already defined in the kernel

Syntax

MATLAB command line

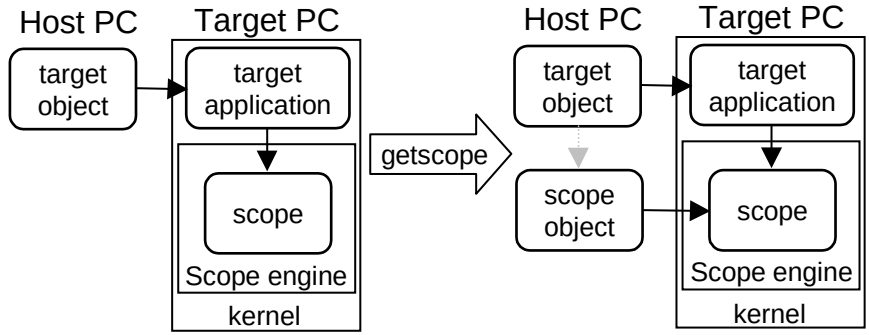
```
scope _object_vector = getscope(target_object, scope_index)

scope_object_vector = target_object.getscope(scope_index)
```

Arguments

target_object	Name or a target object.
scope_index_vector	Vector of existing scope indices listed in the target property Scopes. The vector may have only one element.
scope_object_vector	MATLAB variable for a new scope object vector. The vector many have only one scope object.

Description

Method of a target object. Returns a scope object vector. If you try to get an nonexistent scope, the result is an error. The list of existing scopes may be retrieved using the method get(target_object, 'scopes') or target_object.scopes.



Examples

If your Simulink model has an xPC Target scope block, a scope of type target is created at the time the target application is downloaded to the target PC. To change the number of samples, you need to create a scope object and then change the scope object property NumSamples.

```
sc1 = getscope(tg,1)  or sc1 = tg.getscope(1)
sc1.NumSample = 500
```

To get the properties of all scopes on the target PC and create a vector of scope objects on the host PC. If the target object has more than one scope, creates a vector or scope objects.

```
scvector = getscope(tg)
```

See Also

The xPC Target target object methods `addscope` and `remscope`. The xPC target M-file demo scripts listed in “xPC Target Demos” on page 5-21.

getsignalid

Purpose

Get the signal index or property name from the signal list

Syntax

MATLAB command line

```
getsignalid(target_object, 'block_name')
getsignalid(target_object, 'block_name', 'flag')
```

Arguments

`target_object`	Name of an existing target object.
`block_name`	Name of a Simulink block from you model.
flag	If flag = property, then return the property name for the signal. If flag = numeric, then return a number index. This argument is optional with the default behavior to return a number.

Description

Method of a target object. Returns the index or name of a signal from the signal list, based on the path to the signal name. The block names must be entered in full and are case-sensitive.

Examples

Get the property name for the parameter Gain in the Simulink block Gain1.

```
getsignalid(tg, 'Gain1') or tg.getsignal('Gain1')
ans = S6
```

Get the property index for the parameter Gain in the Simulink block Gain1.

```
getsignalid(tg, 'Gain1', 'Gain', 'numeric')
ans = 6
```

S6 is a property of the target object. For example, you could get the value of a signal with `signal_6 = tg.s6`.

See Also

The target object method `getparamid`. The xPC target M-file demo scripts listed in "xPC Target Demos" on page 5-21.

Purpose

Download a target application to the target PC

Syntax

MATLAB command line

```
load(target_object,'target_application')
target_object.load('target_application')
```

Arguments

target_object	Name of an existing target object
target_application	Simulink model and target application name.

Description

Method of a target object. Before using this function, the target PC must be booted with the xPC Target kernel, and the target application must be built in the current working directory on the host PC.

If an application was previously loaded, the old target application is first unloaded before downloading the new target application. The method `load` is called automatically after the RTW build process.

Examples

Load the target application `xpcosc` represented by the target object `tg`.

```
load(tg,'xpcosc') or tg.load('xpcosc')
+tg or tg.start or start(tg)
```

See Also

The xPC Target function `unload`. The xPC target M-file demo scripts listed in "xPC Target Demos" on page 5-21.

reboot

Purpose

Reboot the target PC

Syntax

MATLAB command line

```
reboot(target_object)
```

Target PC command line

```
reboot
```

Arguments

`target_object`	Name of an existing target object

Description

Method of a target object. Reboots the target PC, and if a target boot disk is still present, the xPC target kernal is reloaded.

You can also use this method to reboot the target PC back to Windows after removing the target boot disk.

Note This method may not work on some target hardware.

See Also

The xPC Target target object methods `load` and `unload`.

Purpose

Remove a scope from the target PC.

Syntax

MATLAB command line

```
remscope(target_object, scope_index_vector)
target_object.remscope(scope_index_vector)

remscope(target_object)
target_object.remscope
```

Target PC command line

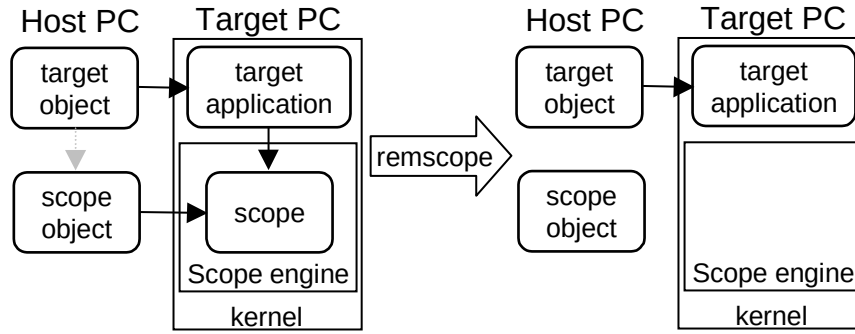
```
remscope scope_index
remscope 'all'
```

Arguments

`target_object`	Name of a target object. The default name it `tg`.
`scope_index_vector`	Vector of existing scope indices listed in the target property Scopes.
scope_index	Single scope index.

Description

Method of a target object. If a scope index is not given, then the method remscope deletes all scopes on the target PC. The method `remscope` has no return value. The scope object representing the scope on the host PC is not deleted.



Examples

Remove a single scope.

```
remscope(tg,1) or tg.remscope(1)
```

Remove two scopes.

```
remscope(tg,[1 2]) or tg.remscope([1,2])
```

Remove all scopes.

```
remscope(tg) or tg.remscope
```

See Also

The xPC Target target object methods addscope and getscope. The xPC target M-file demo scripts listed in “xPC Target Demos” on page 5-21.

Purpose

Change property values for target objects

Syntax

MATLAB command line

```
set(target_object)

set(target_object, property_name1, property_value1,
property_name2, property_value2, . . .)

target_object.set('property_name1', property_value1)

set(target_object, property_name_vector, property_value_vector)

target_object_name.property_name = property_value
```

Target PC command line - Commands are limited to the target object properties: stoptime, sampletime, and parameters.

```
parameter_name = parameter_value
stoptime = floating_point_number
sampletime = floating_point_number
```

Arguments

`target_object`	Name of a target object
`property_name`	Name of a scope object property. Always use quotes
`property_value`	Value for a scope object property. Always use quotes for character strings, quotes are optional for numbers.
`parameter_name`	The letter p followed by the parameter index. For example, `p0`, `p1`, `p2`.

Description

Method of a target object. Sets the properties of the target object. Not all properties are user-writable.

Properties must be entered in pairs, or using the alternate syntax, as one-dimensional cell arrays of the same size. This means they have to both be row vectors or both column vectors, and the corresponding values for properties in property_name_vector are stored in property_value_vector.

The function set typically does not return a value. However, if called with an explicit return argument, for example, a = set(target_object, property_name,

property_value), it returns the value of the properties after the indicated settings have been made.

Examples

Get a list of writable properties for a scope object.

```
sc1 = getscope(tg,1)
set(sc1)
```

```
xPC Target Object:
    Writable Properties

    StopTime
    SampleTime
    ViewMode
    LogMode             : [0 | 1]
    ShowParameters      : [On | {Off}]
    ShowSignals         : [On | {Off}]
```

Change the property showsignals to on.

```
tg.set('showsignals', 'on') or set(tg, 'showsignals', 'on')
```

As an alternative to the method `set`, use the target object property `showsignals`. In the MATLAB window, type

```
tg.showsignals ='on'
```

See Also

The xPC Target target object methods `get`. The scope object methods `get` and `set`. The built in MATLAB functions `get` and `set`. The xPC target M-file demo scripts listed in "xPC Target Demos" on page 5-21.

Purpose

Start execution of a target application on a target PC.

Syntax

MATLAB command line

```
start(target_object)
target_object.start
+target_object
```

Target PC command line

```
start
```

Arguments

`target_object`	Name of a target object. The default name is `tg`.

Description

Method of both target and scope objects. Starts execution of the target application represented by the target object. Before using this method, the target application must be created and loaded on the target PC. If a target application is running, this command has no effect.

Examples

Start the target application represented by the target object tg.

`+tg` or `tg.start` or `start(tg)`

See Also

The xPC Target target object methods `stop` on page 6-36, `load` on page 6-29, and `unload` on page 6-37. The scope object method `stop` on page 7-20.

stop

Purpose

Stop execution of a target application on a target PC.

Syntax

MATLAB command line

```
stop(target_object)
target_object.stop
-target_object
```

Target PC command line

```
stop
```

Arguments

`target_object`	Name of a target object.

Description

Stops execution of the target application represented by the target object. If the target application is stopped, this command has no effect.

Examples

Stop the target application represented by the target object tg.

```
stop(tg) or tg.stop or -tg
```

See Also

The xPC Target target object method `start` on page 6-35. The scope object methods `stop` on page 7-20 and `start` on page 7-18.

Purpose

Removes the current target application from the target PC.

Syntax

MATLAB command line

```
unload(target_object)
target_object.unload
```

Arguments

`target_object`	Name of a target object that represents a target application.

Description

Method of a target object. The kernel goes into loader mode is ready to download new target application from the host PC.

Examples

Unload the target application represented by the target object `tg`.

```
unload(tg) or tg.unload
```

See Also

The xPC Target methods load and reboot.

xpc

Purpose

Create a target object representing the target application

Syntax

MATLAB command line

```
target_object = xpc
```

Arguments

`target_object`	Variable name to reference the target object.

Description

Constructor of a target object. The target object represents the target application and target PC. Changes are made to the target application by making changes to the target object using methods and properties.

Examples

Before you build a target application, you can check the connection between your host and target computers by creating a target object.

```
tg = xpc
```

```
xPC Object

   Connected            = Yes
   Application          = loader
```

See Also

The xPC Target methods `get` on page 6-22, `set` on page 6-33.

7

Scope Object Reference

Use scope objects to run and control scopes on the target PC.

This chapter includes the following sections:

- **"Scope Object"** — Definition, properties, and methods
- **"Using Scope Objects"** — Changing properties, and running methods

Scope Object

xPC Target uses scopes and scope objects as an alternative to using Simulink scopes and external mode. Understanding the structure of scope objects will help you to develop a mental model of the xPC Target software environment.

This section includes the following topics:

- **"What is a Scope Object?"** — Definition, and ways to create scope objects
- **"Scope Object Properties"** — List of properties with definitions
- **"Scope Object Methods"** — List of methods with definitions

What is a Scope Object?

A scope object on the host PC represents a scope on the target PC. You use scope objects to observe the signals from your target application during a real-time run or analyze the data after the run is finished.

To create a scope object:

- Add an xPC Target scope block to your Simulink model, build the model to create a scope, and then use the target object method `getscope` to create a scope object.
- Use the target object method `addscope` to create a scope, create a scope object and assign the scope properties to the scope object.

A scope object has associated properties and methods specific to that object.

Scope Object Properties

Scope object properties let you select signals to acquire, set triggering modes, and access signal information from the target application. You can view and change these properties using scope object methods

The properties for a scope object are listed in the following table. This table includes a description of the properties and which properties you can change directly by assigning a value.

Table 7-1: List of Scope Object Properties

Property	Description	Write
`Application`	Name of the Simulink model associated to this scope object.	
`ScopeId`	A numeric index unique for each scope.	
`Status`	Indicates whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are `'Acquiring'`, `'Ready for being Triggered'`, `'Interrupted'`, and `'Finished'`.	
`Type`	Determines whether the scope is displayed on the host computer or on the target computer. Values are `'host'` and `'target'`.	
`NumSamples`	Number of contiguous samples captured during the acquisition of a data package.	Yes
`NumPrePostSamples`	Number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set **TriggerMode** to FreeRun, this property has no effect on data acquisition.	

Table 7-1: List of Scope Object Properties

Property	Description	Write
`Decimation`	A number `n`, where every nth sample is acquired in a scope window. **Note** This value is the same as **Interleave** in a scope window.	Yes
`TriggerMode`	Trigger mode for a scope. Valid values are `'FreeRun'` (default), `'Software'`, `'Signal'`, and `'Scope'`.	Yes
`TriggerSignal`	If `TriggerMode='Signal'`, identifies which block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property Signal.	Yes
`TriggerLevel`	If `TriggerMode='Signal'`, indicates the value the signal has to cross to trigger the scope and start acquiring data. The trigger level can be crossed with either a rising or falling signal.	Yes
`TriggerSlope`	If `TriggerMode='Signal'`, indicates whether the trigger in on a rising or falling signal. Values are `'Either'` (default), `'Rising'`, or `'Falling'`.	Yes
`TriggerScope`	If `TriggerMode='Scope'`, identifies which scope to use for a trigger. A scope can be set to trigger when another scope is triggered. This is done by setting the slave scope property `TriggerScope` to the scope index of the master scope.	Yes
`Mode`	Indicates how a scope displays the signals. Values are `'Numerical'`, `'Redraw'` (default), `'Sliding'`, or `'Rolling'`.	Yes

Table 7-1: List of Scope Object Properties

Property	Description	Write
`YLimit`	Minimum and maximum y-axis values. This property can be set to `'auto'`.	Yes
`Grid`	Values are `'on'` or `'off'`.	Yes
`StartTime`	Time within the total execution time, when a scope begins acquiring a data package.	
`Time`	Contains the time data for a single data package from a scope.	
`Data`	Contains the output data for a single data package from a scope.	
`Signals`	List of signal indices from the target object to display on the scope.	

Scope Object Methods

The scope object methods allow you to control scopes on your target PC. The methods are listed in the following table.

Table 7-2: List of Scope Object Methods

Scope Method	Description
`set`	Sets writable scope object properties to the specified value. For a list of writable values, use `set(scope_object)`
`get`	Returns the value of readable properties from a scope object.
`addsignal`	Adds a signal to a scope and a scope object. The signal is specified using the signal indices from the target object.

Table 7-2: List of Scope Object Methods

Scope Method	Description
`remsignal`	Removes a signal from a scope and a scope object. The signal is specified using signal indices from the scope object.
`start`	Starts a scope, but does not necessarily start the acquisition of data. The acquisition of data is dependent on the trigger mode.
`stop`	Stops a scope and the acquisition of a data.
`trigger`	If `TriggerMode='Software'`, starts the acquisition of data from the target application.

Using Scope Objects

xPC Target uses scope objects to represent scopes on the target PC. This section shows some of the common tasks that you use with scope objects.

This section includes the following topics:

- "Displaying Scope Object Properties for a Single Scope"
- "Displaying Scope Object Properties for All Scopes"
- "Setting the Value of a Scope Property"
- "Getting the Value of a Scope Property"
- "Using the Method Syntax with Scope Objects"

Displaying Scope Object Properties for a Single Scope

To list the properties of a single scope object `sc1`:

1 In the MATLAB window, type

```
sc1  = getscope(tg,1) or sc1 = tg.getscopes(1)
```

MATLAB creates the scope object `sc1` from a previously created scope.

2 Type

```
sc1
```

The current scope properties are uploaded to the host PC, and then MATLAB displays a list of the scope object properties with the updated values. Because `sc1` is a vector with a single element, you could also type, `sc1(1)` or `sc1([1])`.

Note Only scopes with type `host` store data in the properties scope_object.Time and scope_object.Data.

For a list of target object properties with a description, see "Target Object Properties" on page 6-4.

Displaying Scope Object Properties for All Scopes

To list the properties of all scope objects associated with the target object `tg`:

1. In the MATLAB window, type

   ```
   getscope(tg) or tg.getscope
   ```

   MATLAB displays a list of all scope objects associated with the target object.

2. Alternately, type

   ```
   allscopes = getscope(tg) or allscopes = tg.getscope
   allscopes
   ```

   The current scope properties are uploaded to the host PC, and then MATLAB displays a list of all the scope object properties with the updated values. To list some of the scopes, use the vector index. For example, to list the fist and third scopes, type `allscopes([1,3])`.

For a list of target object properties with a description, see "Target Object Properties" on page 6-4.

Setting the Value of a Scope Property

With xPC Target you can use either a function syntax or an object property syntax. The syntax `set(scope_object, property_name, new_property_value)` can be replaced by:

- `scope_object_vector.property_name = new_property_value.`
- `scope_object(index_vector).propety_name = new_property_value.`

For example to change the trigger mode for the scope object `sc1`:

1. In the MATLAB window, type

   ```
   sc1.triggermode = 'signal'
   ```

2. Alternately, you could type

   ```
   set(sc1,'triggermode', 'signal') or sc1.set('triggermode', 'signal')
   ```

Assignment for may also be done for a vector of scope objects, for example allscopes([1, 2]).numsamples = 500. Notice, the indices are MATLAB vector indices and not xPC Target scope indices.

To get a list of the writable properties, type `set(scope_object)`.

Note Method names are case-sensitive, but property names are not.

Getting the Value of a Scope Property

You can list a property value in the MATLAB window, or assign that value to a MATLAB variable. With xPC Target you can use either a function syntax or an object property syntax.

The syntax `get(scope_object_vector, property_name)` can be replaced by

- `scope_object_vector.property_name`
- `scope_object_vector(index_vector).property_name`

For example to assign the start time from the scope object `sc1`:

1. In the MATLAB window, type

   ```
   beginrun = sc1.starttime
   ```

2. Alternately, you could type

   ```
   beginrun = get(sc1,'starttime') or sc1.get('starttime')
   ```

Assignment may also be done using a vector of scope objects, for example scopetypes = allscopes([1, 2]).type. Notice, the indices are MATLAB vector indices and not xPC Target scope indices.

To get a list of readable properties, type `scope_object`. Without assigning to a variable, the property values are listed in the MATLAB window.

Note Method names are case-sensitive, but property name are not.

Using the Method Syntax with Scope Objects

Use the method syntax to run a scope object method. The syntax `method_name(scope_object_vector, argument_list)` can be replaced with:

- `scope_object.method_name(argument_list)`
- `scope_object_vector(index_vector).method_name(list of arguments)`

Unlike properties, for which partial but unambiguous names are permitted, method names must be entered in full, and in lowercase. For example, to add signals to the first scope in a vector of all scopes:

1. In the MATLAB window, type

   ```
   allscopes(1).addsignal([0,1])
   ```

2. Alternately, you could type

   ```
   addsignal(allscopes(1), [0,1])
   ```

addsignal

Purpose

Adds signals to a scope represented by a scope object

Syntax

MATLAB command line

```
addsignal(scope_object_vector, signal_index_vector)

scope_object_vector.addsignal(signal_index_vector)
```

Target command line

```
addsignal scope_index = signal_index, signal_index, . . .
```

Arguments

`scope_object_vector`	Name of a single scope object, or the name of a vector of scope objects.
`signal_index_vector`	For one signal, use a single number. For two or more signals, enclose numbers in brackets and separate with commas.
`scope_index`	Single scope index.

Description

Method of a scope object. The signals must be specified by their index, which may be retrieve using the target object method `getsignalid`. If the scope_object_vector has two or more scope objects, the same signals are assigned to each scope.

Examples

Add signals `0` and `1` from the target object `tg` to the scope object sc1. The signals are added to the scope, and the scope object property Signals is updated to include the added signals.

```
sc1 = getscope(tg,1)
addsignal(sc1,[0,1]) or sc1.addsignal([0,1])
```

Display a list of properties and values for the scope object `sc1` with the property Signals shown below.

```
sc1.Signals

Signals              = 1  : Signal Generator
                       0  : Integrator1
```

Other ways to add signals without using the method addsignal is to use the scope object method `set`.

```
set(sc1,'Signals', [0,1]) or sc1.set('signals',[0,1],
```

Or to directly assign signal values to the scope object property Signals.

```
sc1.signals = [0,1].
```

See Also

The xPC Target scope object methods `remsignal` and `set`. The target object method `addscope` and `getsignalid`.

get

Purpose

Return the property values for scope objects

Syntax

MATLAB command line

```
get(scope_object_vector)

get(scope_object_vector, 'scope_object_property')

get(scope_object_vector, scope_object_property_vector)
```

Arguments

target_object	Name of a target object
scope_object_vector	Name of a single scope, or name of a vector of scope objects
scope_object_property	Name of a scope object property

Description

Method of scope objects. Gets the value of readable scope object properties from a scope object or the same property from each scope object in a vector of scope objects.

Examples

List all of the readable properties, along with their present values. This is given in the form of a structure, whose fieldnames are the property names and field values are property values.

```
get(sc)
```

List the value for the scope object property Type. Notice the property name is a string, in quotes, and is not case-sensitive.

```
get(sc,'type')
ans = Target
```

See Also

The xPC Target scope object method set. The target object methods get and set. The built in MATLAB functions get and set.

Purpose

Remove signals from a scope represented by a scope object

Syntax

MATLAB command line

```
remsignal(scope_object)
remsignal(scope_object, signal_index_vector)
scope_object.remsignal(signal_index_vector)
```

Target command line

```
remsignal scope_index = signal_index, signal_index, . . .
```

Arguments

`scope_object`	MATLAB object created with the target object methods addscope or getscope.
`signal_index_vector`	Index numbers from the scope object property `Signals`. This argument is optional and if left out all signals are removed.
`signal_index`	Single signal index.

Description

Method for a scope object. The signals must be specified by their index, which may be retrieved using the target object method `getsignalid`. If the scope_object_vector has two or more scope object, the same signals are removed from each scope. The argument SIGNALS is optional; if left out, all signals are removed.

Examples

Remove signals 0 and 1 from the scope represented by the scope object sc1.

```
sc1.get('signals')

ans= 0 1
```

Removed signals from the scope on the target PC with the scope object property Signals updated.

```
remsignal(sc1,[0,1]) or sc1.remsignal([0,1])
```

See Also

The xPC Target scope object method `remsignal`, and the target object method `getsignalid`.

set

Purpose

Change property values for scope objects

Syntax

MATLAB command line

```
set(scope_object_vector)

set(scope_object_vector, property_name1, property_value1,
property_name2, property_value2, . . .)

scope_object_vector.set('property_name1', property_value1, ..)

set(scope_object, 'property_name', property_valuse, . . .)
```

Arguments

`scope_object`	Name of a scope object, or a vector of scope objects
`property_name`	Name of a scope object property. Always use quotes
`property_value`	Value for a scope object property. Always use quotes for character strings, quotes are optional for numbers.

Description

Method for scope objects. Sets the properties of the scope object. Not all properties are user-writable

Properties must be entered in pairs, or using the alternate syntax, as one-dimensional cell arrays of the same size. This means they have to both be row vectors or both column vectors, and the corresponding values for properties in property_name_vector are stored in property_value_vector.

The function set typically does not return a value. However, if called with an explicit return argument, for example, a = set(target_object, property_name, property_value), it returns the value of the properties after the indicated settings have been made.

Examples

Get a list of writable properties for a scope object.

```
sc1 = getscope(tg,1)
set(sc1)

xPC Scope Object:
    Writable Properties
```

```
NumSamples
Decimation
TriggerMode   : [{FreeRun} | Software | Signal | Scope]
TriggerSignal
TriggerLevel
TriggerSlope  : [{Either} | Rising | Falling]
TriggerScope
Signals
Mode          : [Numerical | {Redraw} | Sliding | Rolling]
YLimit
Grid
```

The property value for the scope object sc1 is changed to on

```
sc1.set('grid', 'on') or set(sc1, 'grid', 'on')
```

See Also

The xPC Target scope object method get. The target object methods set and get. The built in MATLAB functions get and set.

start

Purpose

Start execution of a scope on a target PC

Syntax

MATLAB command line

```
start(scope_object_vector)
scope_object_vector.start
+scope_object_vector

start(getscope(target_object, scope_index_vector))
```

Target PC command line

```
startscope scope_index
startscope 'all'
```

Arguments

`target_object`	Name of a target object.
`scope_object_vector`	Name of a single scope object, name of vector of scope objects, list of scope object names in a vector form [scope_object1, scope_object2], or the target object method `getscope` which returns a scope_object vector.
`scope_index_vector`	Index for a single scope, or list of scope indices in vector form.
`scope_index`	Single scope index.

Description

Method for scope objects. Starts a scope on the target PC represented by a scope object on the host PC. This method does not necessarily start data acquisition which depends on the trigger settings. Before using this method, a scope must be created. To create a scope, use the target object method `addscope` or add xPC Target scope blocks to your Simulink model.

Examples

Start one scope with the scope object sc1.

```
sc1 = getscope(tg,1) or sc1 = tg.getscope(1)
start(sc1) or sc1.start or +sc1
```

or type

```
start(getscope(tg,1))
```

Start two scopes.

```
somescopes = getscope(tg,[1,2]) or somescopes=
tg.getscope([1,2])
start(somescopes) or somescopes.start
```

or type

```
sc1 = getscope(tg,1) or sc1 =tg.getscope(1)
sc2 = getscope(tg,2) or sc2 = tg.getscope(2)
start([sc1,sc2])
```

or type

```
start(getscope(tg,[1,2])
```

Start all scopes

```
allscopes = getscope(tg) or allscopes = tg.getscope
start(allscopes) or allscopes.start or +allscopes
```

or type

```
start(getscope(tg)) or start(tg.getscope)
```

See Also

The xPC Target target object methods `getscope` and `stop`. The scope object method `stop`.

stop

Purpose

Stop execution of a scope on the target PC.

Syntax

MATLAB command line

```
stop(scope_object_vector)
scope_object.stop
-scope_object

stop(getscope(target_object, scope_index_vector))
```

Target PC command line

```
stopscope scope_index
stopscope 'all'
```

Arguments

`target_object`	Name of a target object.
`scope_object_vector`	Name of a single scope object, name of vector of scope objects, list of scope object names in a vector form [scope_object1, scope_object2], or the target object method `getscope` which returns a scope_object vector.
`scope_index_vector`	Index for a single scope, or list of scope indices in vector form.
`scope_index`	Single scope index.

Description

Method for a scope objects. Stops the scopes represented by the scope objects.

Examples

Stop one scope represented by the scope object sc1.

```
stop(sc1) or sc1.stop or -sc1
```

Stop all scopes with a scope object vector allscopes created with the command allscopes = getscope(tg) or allscopes = tg.getscope.

```
stop(allscopes) or allscopes.stop or -allscopes
```

or type

```
stop(getscope(tg)) or stop(tg.getscope)
```

See Also

The xPC Target target object methods `getscope`, `stop`, and `start`. The scope object method `start`.

trigger

Purpose

Software trigger the start of data acquisition for one or more scopes.

Syntax

MATLAB command line

```
trigger(scope_object_vector) or scope_object_vector.trigger
```

Arguments

`scope_object_vector`	Name of a single scope object, name of a vector of scope objects, list of scope object names in a vector form [scope_object1, scope_object2], or the target object method `getscope` which returns a scope_object vector.

Description

Method for a scope object. If the scope object property TriggerMode has a value of 'software', then this function triggers the scope represented by the scope object to acquire the number of data points in the scope object property `NumSamples`.

Note Only scopes with type `host` store data in the properties scope_object.Time and scope_object.Data.

Examples

Set a single scope to software trigger, trigger the acquisition or one set of samples, nd plot data.

```
sc1 = tg.addscope('host',1) or sc1=addscope(tg,'host',1)
sc1.triggermode = 'software'

tg.start, or start(tg), or +tg
sc1.start  or start(sc1)  or +sc1
sc1.trigger  or trigger(sc1)

plot(sc1.time, sc1.data)

sc1.stop or stop(sc1) or -sc1
tg.stop or stop(tg) or -tg1
```

Set all scopes to software trigger and trigger to start.

```
allscopes = tg.getscopes
allscopes.triggermode = 'software'
allscopes.start  or start(allscopes) or +allscopes
allscopes.trigger  or trigger(allscopes)
```

Index