

**xPC Target™**

User's Guide

**R2012a**

**MATLAB®  
& SIMULINK®**

## How to Contact MathWorks



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



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



508-647-7000 (Phone)



508-647-7001 (Fax)



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

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

*xPC Target™ User's Guide*

© COPYRIGHT 1999–2012 by The MathWorks, Inc.

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

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

### Trademarks

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

### Patents

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

## Revision History

September 1999	First printing	New for Version 1 (Release 11.1)
November 2000	Online only	Revised for Version 1.1 (Release 12)
June 2001	Online only	Revised for Version 1.2 (Release 12.1)
September 2001	Online only	Revised for Version 1.3 (Release 12.1+)
July 2002	Online only	Revised for Version 2 (Release 13)
June 2004	Online only	Revised for Version 2.5 (Release 14)
August 2004	Online only	Revised for Version 2.6 (Release 14+)
October 2004	Online only	Revised for Version 2.6.1 (Release 14SP1)
November 2004	Online only	Revised for Version 2.7 (Release 14SP1+)
March 2005	Online only	Revised for Version 2.7.2 (Release 14SP2)
September 2005	Online only	Revised for Version 2.8 (Release 14SP3)
March 2006	Online only	Revised for Version 2.9 (Release 2006a)
May 2006	Online only	Revised for Version 3.0 (Release 2006a+)
September 2006	Online only	Revised for Version 3.1 (Release 2006b)
March 2007	Online only	Revised for Version 3.2 (Release 2007a)
September 2007	Online only	Revised for Version 3.3 (Release 2007b)
March 2008	Online only	Revised for Version 3.4 (Release 2008a)
October 2008	Online only	Revised for Version 4.0 (Release 2008b)
March 2009	Online only	Revised for Version 4.1 (Release 2009a)
September 2009	Online only	Revised for Version 4.2 (Release 2009b)
March 2010	Online only	Revised for Version 4.3 (Release 2010a)
September 2010	Online only	Revised for Version 4.4 (Release 2010b)
April 2011	Online only	Revised for Version 5.0 (Release 2011a)
September 2011	Online only	Revised for Version 5.1 (Release 2011b)
March 2012	Online only	Revised for Version 5.2 (Release 2012a)



## Model Architectures

### FPGA Models

# 1

<b>FPGA Support</b> .....	1-2
Supported FPGA I/O Boards .....	1-2
Prerequisites .....	1-3
<b>Workflow</b> .....	1-4
Creating an FPGA Domain Model .....	1-6
Generating HDL with the Workflow Advisor .....	1-7
Creating an xPC Target Domain Model .....	1-13
Adding the xPC Target Interface Subsystem to the xPC Target Domain Model .....	1-14
Building and Downloading the Target Application .....	1-16
Interrupts .....	1-17
<b>Demos</b> .....	1-20

### Vector CANape Support

# 2

<b>Vector CANape</b> .....	2-2
Introduction .....	2-2
xPC Target and Vector CANape Limitations .....	2-3
<b>Configuring the xPC Target and Vector CANape</b>	
<b>Software</b> .....	2-4
Setting Up and Building the Model .....	2-4
Creating a New Vector CANape Project to Associate with a Particular Target Application .....	2-6
Configuring the Vector CANape Device .....	2-7

Providing A2L (ASAP2) Files for the Vector CANape Database .....	2-10
<b>Event Mode Data Acquisition .....</b>	<b>2-11</b>
Guidelines .....	2-11
Limitations .....	2-11

## Incorporating Fortran S-Functions

### 3

<b>Fortran S-Functions .....</b>	<b>3-2</b>
Prerequisites .....	3-2
Simulink Demos Folder .....	3-2
Steps to Incorporate Fortran .....	3-3
 <b>Atmosphere Model Example .....</b>	 <b>3-5</b>
In This Example .....	3-5
Creating a Fortran Atmosphere Model .....	3-5
Compiling Fortran Files .....	3-7
Creating a C-MEX Wrapper S-Function .....	3-8
Compiling and Linking the Wrapper S-Function .....	3-12
Validating the Fortran Code and Wrapper S-Function ...	3-14
Preparing the Model for the xPC Target Application	
Build .....	3-14
Building and Running the xPC Target Application .....	3-16

## Target Application Environment

### 4

<b>xPC Target Options Configuration Parameter .....</b>	<b>4-2</b>
 <b>xPC Target Explorer .....</b>	 <b>4-3</b>
Basic Operations .....	4-3
Menus, Toolbars, and Shortcut Keys .....	4-6
Default Target Computers .....	4-8

<b>Target Environment Command-Line Interface</b> .....	<b>4-10</b>
Creating Target PC Environment Object Containers .....	<b>4-10</b>
Displaying Target PC Environment Object Property Values .....	<b>4-10</b>
Adding Target PC Environment Collection Objects .....	<b>4-11</b>
Removing Target PC Environment Collection Objects .....	<b>4-11</b>
Getting Target PC Environment Object Names .....	<b>4-11</b>
Changing Target PC Environment Object Defaults .....	<b>4-12</b>
Working with Particular Target PC Object Environments .....	<b>4-12</b>
<b>Setting Up the Target Application Environment</b> .....	<b>4-14</b>
Introduction .....	<b>4-14</b>
Getting a List of Environment Properties .....	<b>4-14</b>
Getting a List of Environment Properties for Single target computer Systems .....	<b>4-15</b>
Changing Environment Properties with xPC Target Explorer .....	<b>4-16</b>
Changing Environment Properties with a Command-Line Interface .....	<b>4-19</b>
Changing Environment Properties with a Command-Line Interface for Single Target Computer Systems .....	<b>4-20</b>
<b>Configuring Environment From the MATLAB Command Line</b> .....	<b>4-21</b>
Setup Requirements .....	<b>4-21</b>
Configuring the Compiler .....	<b>4-22</b>
Configuring Communications .....	<b>4-23</b>
Configuring Boot Methods .....	<b>4-25</b>
<b>Exporting and Importing Environment Properties</b> ...	<b>4-31</b>

## Signals and Parameters

# 5

<b>Signal Monitoring</b> .....	<b>5-2</b>
Introduction .....	<b>5-2</b>
Signal Monitoring with xPC Target Explorer .....	<b>5-2</b>
Signal Monitoring with the MATLAB Interface .....	<b>5-9</b>

Monitoring Stateflow States .....	5-10
Animating Stateflow Charts .....	5-14
<b>Signal Tracing</b> .....	<b>5-16</b>
Introduction .....	5-16
Signal Tracing with xPC Target Explorer .....	5-16
Signal Tracing with the MATLAB Interface .....	5-40
Signal Tracing with xPC Target Scope Blocks .....	5-49
Signal Tracing with Simulink External Mode .....	5-51
Signal Tracing with a Web Browser .....	5-55
<b>Signal Logging</b> .....	<b>5-57</b>
Introduction .....	5-57
Signal Logging with xPC Target Explorer .....	5-57
Signal Logging in the MATLAB Interface .....	5-60
Signal Logging with a Web Browser .....	5-64
<b>Parameter Tuning and Inlining Parameters</b> .....	<b>5-66</b>
Introduction .....	5-66
Parameter Tuning with xPC Target Explorer .....	5-66
Parameter Tuning with the MATLAB Interface .....	5-70
Parameter Tuning with Simulink External Mode .....	5-73
Parameter Tuning with a Web Browser .....	5-75
Saving and Reloading Application Parameters with the MATLAB Interface .....	5-76
Inlined Parameters .....	5-79
<b>Nonobservable Signals and Parameters</b> .....	<b>5-86</b>

## Execution Modes

# 6

<b>Introducing Execution Modes</b> .....	<b>6-2</b>
Introduction .....	6-2
<b>Interrupt Mode</b> .....	<b>6-3</b>
Latencies Introduced by Interrupt Mode .....	6-3



<b>Polling Mode</b> .....	<b>6-5</b>
Introducing Polling Mode .....	<b>6-5</b>
Setting the Polling Mode .....	<b>6-7</b>
Restrictions Introduced by Polling Mode .....	<b>6-11</b>
Controlling the Target Application .....	<b>6-14</b>
Polling Mode Performance .....	<b>6-15</b>
Polling Mode and Multicore Processors .....	<b>6-16</b>

# Execution Using MATLAB® Scripts

## Targets and Scopes in the MATLAB Interface

### 7

<b>Target Driver Objects</b> .....	<b>7-2</b>
What Is a Target Object? .....	<b>7-2</b>
Accessing Help for Target Objects .....	<b>7-3</b>
Creating Target Objects .....	<b>7-3</b>
Displaying Target Object Properties .....	<b>7-4</b>
Setting Target Object Properties from the Host Computer .....	<b>7-5</b>
Getting the Value of a Target Object Property .....	<b>7-6</b>
Using the Method Syntax with Target Objects .....	<b>7-7</b>
 <b>Target Scope Objects</b> .....	 <b>7-8</b>
What Is a Scope Object? .....	<b>7-8</b>
Accessing Help for Scope Objects .....	<b>7-10</b>
Displaying Scope Object Properties for a Single Scope .....	<b>7-10</b>
Displaying Scope Object Properties for All Scopes .....	<b>7-11</b>
Setting the Value of a Scope Property .....	<b>7-11</b>
Getting the Value of a Scope Property .....	<b>7-12</b>
Using the Method Syntax with Scope Objects .....	<b>7-13</b>
Acquiring Signal Data with File Scopes .....	<b>7-14</b>
Acquiring Signal Data into Multiple, Dynamically Named Files with File Scopes .....	<b>7-15</b>
Advanced Data Acquisition Topics .....	<b>7-17</b>

## Logging Signal Data with FTP and File System Objects

### 8

<b>File Systems</b> .....	8-2
<b>FTP and File System Objects</b> .....	8-4
<b>Using xpctarget.ftp Objects</b> .....	8-5
Overview .....	8-5
Accessing Files on a Specific Target Computer .....	8-6
Listing the Contents of the Target Computer Folder .....	8-7
Retrieving a File from the Target Computer to the Host Computer .....	8-8
Copying a File from the Host Computer to the Target Computer .....	8-8
<b>Using xpctarget.fs Objects</b> .....	8-10
Overview .....	8-10
Accessing File Systems from a Specific Target Computer ..	8-11
Retrieving the Contents of a File from the Target Computer to the Host Computer .....	8-12
Removing a File from the Target Computer .....	8-15
Getting a List of Open Files on the Target Computer ....	8-16
Getting Information about a File on the Target Computer .....	8-17
Getting Information about a Disk on the Target Computer .....	8-18

## Execution Using Graphical User Interface Models

### 9

<b>xPC Target Interface Blocks to Simulink Models</b> .....	9-2
Simulink User Interface Model .....	9-2
Creating a Custom Graphical Interface .....	9-3
To xPC Target Block .....	9-4
From xPC Target Block .....	9-5

Creating a Target Application Model .....	9-7
Marking Block Parameters .....	9-7
Marking Block Signals .....	9-10

## Execution Using the Target Computer Command Line

# 10

<b>Target Computer Command-Line Interface .....</b>	<b>10-2</b>
Using Target Application Methods on the Target	
Computer .....	10-2
Manipulating Target Object Properties from the Target	
Computer .....	10-3
Manipulating Scope Objects from the Target Computer ..	10-4
Manipulating Scope Object Properties from the Target	
Computer .....	10-6
Aliasing with Variable Commands on the Target	
Computer .....	10-6

## Execution Using the Web Browser Interface

# 11

<b>Web Browser Interface .....</b>	<b>11-2</b>
Introduction .....	11-2
Connecting the Web Interface Through TCP/IP .....	11-2
Connecting the Web Interface Through RS-232 .....	11-3
Using the Main Pane .....	11-7
Changing WWW Properties .....	11-9
Viewing Signals with a Web Browser .....	11-10
Viewing Parameters with a Web Browser .....	11-11
Changing Access Levels to the Web Browser .....	11-11

# Troubleshooting

## Basic Troubleshooting

### 12

Basic Troubleshooting .....	12-2
-----------------------------	------

## Confidence Test Failures

### 13

Test 1, Ping target PC 'TargetPC1' using system ping: FAILED .....	13-2
Test 2, Ping target PC 'TargetPC1' using xpctargetping: FAILED .....	13-5
Test 3, Software reboot the target PC: FAILED .....	13-7
Test 4, Build and download an xPC Target application using model xpcosc: FAILED .....	13-9
Test 5, Check host-target command communications: FAILED .....	13-12
Test 6, Download a pre-built xPC Target application: FAILED .....	13-14
Test 7, Execute xPC Target application for 0.2s: FAILED .....	13-15
Test 8, Upload logged data and compare with simulation results: FAILED .....	13-16

## Host Computer Configuration

---

### 14

Why Does Boot Drive Creation Halt Without Finishing? .....	14-2
--	------

## Target Computer Configuration

---

### 15

Is the Target Computer BIOS Set as Required? .....	15-2
Can the Target Computer Hard Drive Contain Multiple Partitions? .....	15-3
Why Do I Get a File System Disabled Error? .....	15-4
How Can I Adjust the Stack Size on My Target Computer? .....	15-5
How Can I Obtain PCI Board Information for My Target Computer? .....	15-6
What Do I Do If My I/O Board Does Not Work? .....	15-8

## Host-Target Communication

---

### 16

Is There Communication Between Your Computers? ..	16-2
Are I/O Boards with Slow Initialization Times Causing Communication Problems During Download? .....	16-4

Are Multiple Ethernet Cards in the Target Computer Causing Communication Problems During Download? .....	16-6
Are Errors in I/O Board Drivers Causing Communication Problems During Download? .....	16-8
How Can I Diagnose Network Problems? .....	16-9

## Target Computer Boot Process

---

# 17

Why Is the Target Computer Unable to Boot? .....	17-2
How Do I Fix a Kernel Load Error? .....	17-4
How Do I Fix a Not Bootable Medium Error? .....	17-5
Why Is the Target Computer Halted? .....	17-6

## Modeling

---

# 18

How Do I Handle Register Rollover for Encoder Blocks? .....	18-2
How Can I Write Custom Device Drivers? .....	18-3

## Model Compilation

---

### 19

- How Can I Deploy a Standalone Interface to Control a Target Application? ..... 19-2
- Why Do I Get a Compiler Error When Compiling Models with Links to Dynamic Link Libraries? ..... 19-3
- Why Do I Get a Compiler Error with Some Blocks and Not Others? ..... 19-4

## Application Download

---

### 20

- Why Does My Download Time Out? ..... 20-2
- How Do I Increase the Download Timeout Value? .... 20-4
- Why Does the Download Halt Without Completing? ... 20-5

## Application Execution

---

### 21

- How Can I View the Contents of the Target Computer Display on the Host Computer? ..... 21-2
- Why Is My Requested Sample Time Different from the Measured Sample Time? ..... 21-3
- What Sample Time Can I Expect from a Target Application? ..... 21-5

Why Has the Stop Time Changed? ..... 21-6

## Application Parameters

---

### 22

Why Does the getparamid Function Return  
Nothing? ..... 22-2

Can I Tune All the Parameters in the Model? ..... 22-3

## Application Signals

---

### 23

Why Do I Get Error -10: Invalid File ID on the Target  
Computer? ..... 23-2

Can I Access All the Signals in the Model? ..... 23-3

## Application Performance

---

### 24

How Can I Improve Runtime Performance? ..... 24-2

Why Does Running My Model Cause CPU Overload  
Messages on the Target Computer? ..... 24-4

How Small a Sample Time Can I Use Without CPU  
Overload? ..... 24-6

Can I Allow CPU Overloads? ..... 24-7



## Getting MathWorks Support

### 25

How Do I Find the MathWorks Support Web Site? . . . .	25-2
How Do I Get an Updated Software Release? . . . . .	25-3
What Should I Do After I Get a New Release? . . . . .	25-4
How Do I Contact MathWorks Technical Support? . . . .	25-5

## Tuning Performance

### 26

Building Referenced Models in Parallel . . . . .	26-2
Multicore Processor Configuration . . . . .	26-4
Profiling Target Application Execution . . . . .	26-6
Profiling Overview . . . . .	26-6
Configuring Your Model to Collect Profile Data During Execution . . . . .	26-6
Displaying and Evaluating Profile Data . . . . .	26-7

## Function Reference

### 27

Classes . . . . .	27-2
Target Computers . . . . .	27-3
Target Environments . . . . .	27-4

<b>Target Applications</b> .....	27-5
<b>Scopes</b> .....	27-6
<b>Parameters</b> .....	27-7
<b>Signals</b> .....	27-8
<b>Data Logs</b> .....	27-9
<b>File Systems</b> .....	27-10

## Functions

# 28

## Configuration Parameters

# 29

<b>Setting Configuration Parameters</b> .....	29-2
xPC Target options Pane .....	29-3
Automatically download application after building .....	29-6
Download to default target PC .....	29-7
Specify target PC name .....	29-8
Name of xPC Target object created by build process .....	29-9
Use default communication timeout .....	29-10
Specify the communication timeout in seconds .....	29-11
Execution mode .....	29-12
Real-time interrupt source .....	29-13
I/O board generating the interrupt .....	29-14
PCI slot (-1: autosearch) or ISA base address .....	29-18
Log Task Execution Time .....	29-19
Signal logging data buffer size in doubles .....	29-20
Enable profiling .....	29-22
Number of events (each uses 20 bytes) .....	29-23
Double buffer parameter changes .....	29-24

Load a parameter set from a file on the designated target	
file system .....	<b>29-26</b>
File name .....	<b>29-27</b>
Build COM objects from tagged signals/parameters .....	<b>29-28</b>
Generate CANape extensions .....	<b>29-29</b>
Include model hierarchy on the target application .....	<b>29-30</b>
Enable Stateflow animation .....	<b>29-31</b>

## **Index**

---



# Model Architectures

---

xPC Target™ models are Simulink® Simulink models using special blocks and architectures. For more on xPC Target blocks, see *xPC Target I/O Reference*. For more on constructing custom device driver blocks, see *xPC Target Device Drivers Guide*.

- Chapter 1, “FPGA Models”
- Chapter 2, “Vector CANape Support”
- Chapter 3, “Incorporating Fortran S-Functions”



# FPGA Models

---

- “FPGA Support” on page 1-2
- “Workflow” on page 1-4
- “Demos” on page 1-20

## FPGA Support

### In this section...

“Supported FPGA I/O Boards” on page 1-2

“Prerequisites” on page 1-3

### Supported FPGA I/O Boards

xPC Target and HDL Coder™ software enable you to implement Simulink algorithms and configure I/O functionality on Speedgoat field programmable gate array (FPGA) boards.

Board	Description
Speedgoat IO301	Xilinx® Virtex-II, 6912 logic cells, 64 TTL I/O lines
Speedgoat IO302	Xilinx Virtex-II, 6912 logic cells, 32 RS-422 I/O lines
Speedgoat IO303	Xilinx Virtex-II, 6912 logic cells, 16 TTL and 24 RS-422 I/O lines
Speedgoat IO311	Xilinx Virtex-II, 24192 logic cells, 64 TTL I/O lines
Speedgoat IO312	Xilinx Virtex-II, 24192 logic cells, 32 RS-422 I/O lines
Speedgoat IO313	Xilinx Virtex-II, 24192 logic cells, 16 TTL and 24 RS-422 I/O lines
Speedgoat IO314	Xilinx Virtex-II, 24192 logic cells, 32 LVDS I/O lines
Speedgoat IO325	Xilinx Virtex-4 chip, 41472 logic cells, 64 LVCMOS or 32 LVDS (four are input only) I/O lines, two 16-bit 105 MHz analog input channels

For more information, see “Speedgoat” in the *xPC Target I/O Reference*.



Speedgoat I/O FPGA boards are sold as part of xPC Target Turnkey systems. For xPC Target Turnkey hardware, see <http://www.mathworks.com/products/xpctarget/supported-hardware/index.html>.

## **Prerequisites**

To work with FPGAs in the xPC Target environment, you must:

- Install HDL Coder and Xilinx ISE 10.1. For more information, see in the .
- Install the Speedgoat FPGA I/O board in the target computer.
- Be familiar with FPGA technology. In particular, you must know the clock frequency and the I/O connector pin and channel configuration of your FPGA board.
- Have experience using data type conversion and designing Simulink fixed-point algorithms.

HDL programming experience is not required to generate HDL code for your FPGA target.

## Workflow

### In this section...

“Creating an FPGA Domain Model” on page 1-6

“Generating HDL with the Workflow Advisor” on page 1-7

“Creating an xPC Target Domain Model” on page 1-13

“Adding the xPC Target Interface Subsystem to the xPC Target Domain Model” on page 1-14

“Building and Downloading the Target Application” on page 1-16

“Interrupts” on page 1-17

The general workflow for implementing Simulink algorithms on a Speedgoat FPGA I/O board in a target computer.

- 1** Create an FPGA domain model. This Simulink model contains a subsystem (algorithm) to be programmed onto the FPGA chip.
- 2** From this model, use the HDL Workflow Advisor in HDL Coder to:
  - a** Specify the desired FPGA board.
  - b** Specify the I/O interface.
  - c** Synthesize the Simulink algorithm for FPGA programming.
  - d** Generate an xPC Target interface subsystem.

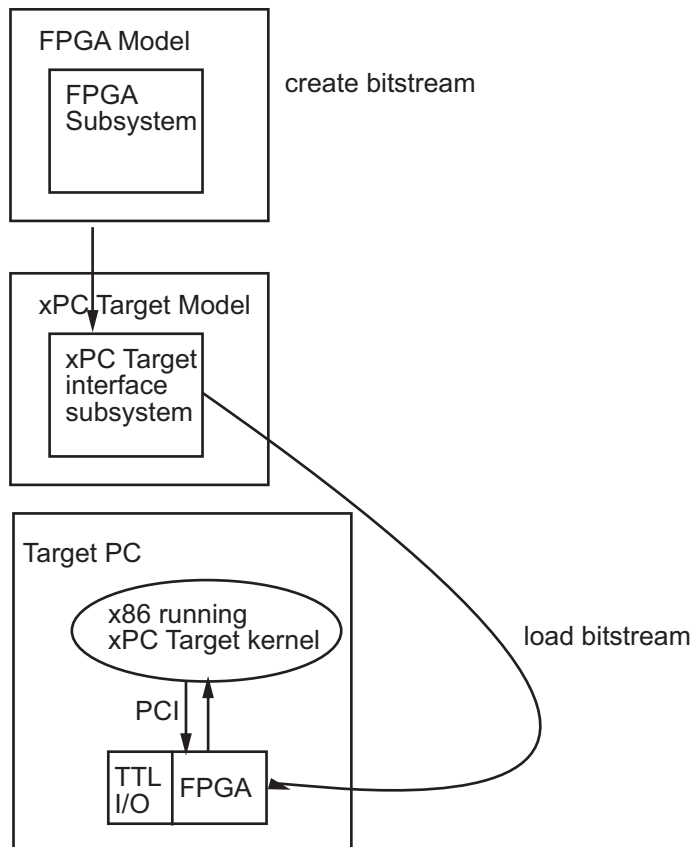
This subsystem contains the blocks that program the FPGA and communicate with the board during download and real-time execution of the target application. For a description of the blocks that the HDL Workflow Advisor incorporates into the subsystem, see “Speedgoat” in the *xPC Target I/O Reference*.

- 3** Create an xPC Target domain model.

This model runs as an xPC Target application on the target computer. The model contains signals that transmit to and receive from the FPGA algorithm through the xPC Target interface subsystem.

- 4** Add the generated xPC Target interface subsystem to the xPC Target domain model. Connect signals to the inports and outputs of the interface subsystem.
- 5** Build the xPC Target domain model and download the application to the target computer. Upon download, the application loads onto the target computer and the algorithm loads onto the FPGA chip. The FPGA algorithm is contained in a bitstream.

The process looks like this.



For an example of this process, see the **Servo Control with the Speedgoat IO301 FPGA Board** demo.

This topic references the demo.

## **Creating an FPGA Domain Model**

A Simulink FPGA model lets you test your FPGA algorithm in a simulation environment before you deploy the algorithm to an FPGA board.

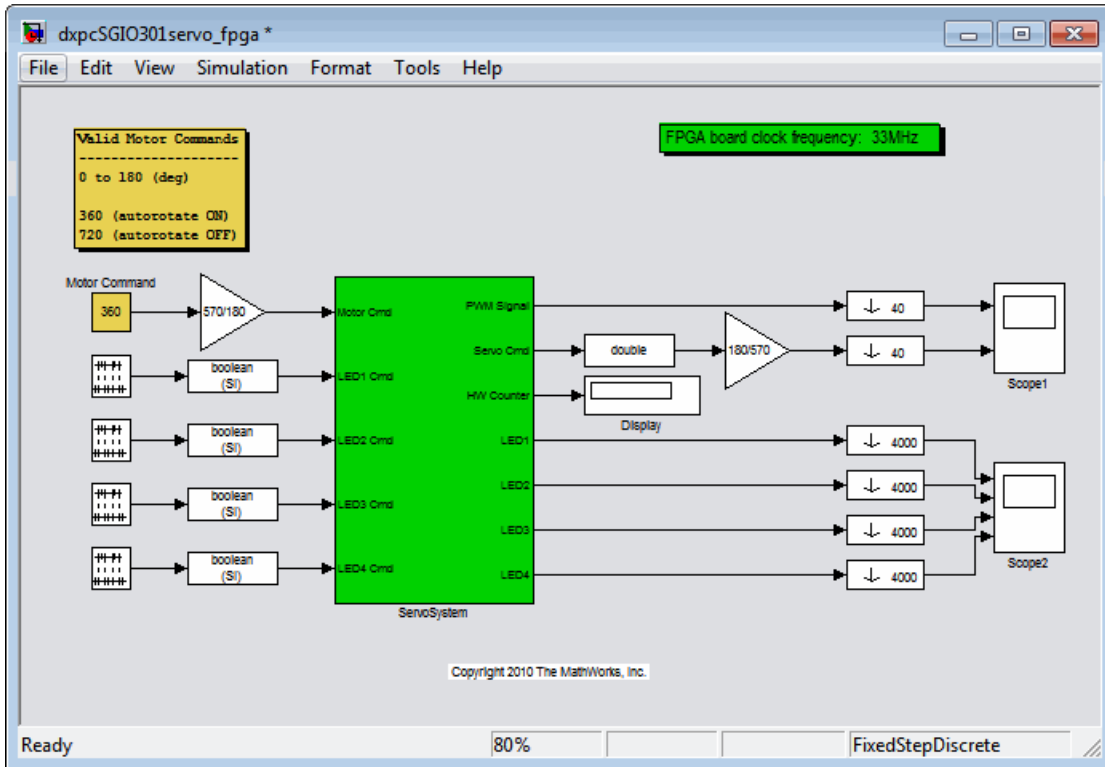
- 1** Create a Simulink model to contain the algorithm that you want to load onto the FPGA.
- 2** Place the algorithm to be programmed on the FPGA inside a Subsystem block. The model can include other blocks and subsystems for testing. However, one subsystem must contain the FPGA algorithm.
- 3** Set or confirm the subsystem inport and outport names and data types.

The HDL Workflow Advisor uses these settings for routing and mapping algorithm signals to I/O connector channels.

- 4** Save the model.

This model is your FPGA domain model. It represents the simulation sample rate of the clock on your FPGA board. For example, the Speedgoat IO301 has an onboard 33MHz clock. One second of simulation equals  $33e6$  iterations of the model.

See `dxpcSGIO301servo_fpga` for an example of an FPGA domain model. The `ServoSystem` subsystem contains the FPGA algorithm.



## Generating HDL with the Workflow Advisor

This topic assumes that you have created an FPGA subsystem (algorithm) in an FPGA domain model. If you have not done so, see “Creating an FPGA Domain Model” on page 1-6.

To generate an HDL code representation of a Simulink subsystem (your FPGA algorithm), use the HDL Workflow Advisor in the HDL Coder to:

- 1 Specify the FPGA board.
- 2 Specify the I/O interface.
- 3 Synthesize the Simulink algorithm for FPGA programming.

- 4 Generate an xPC Target interface subsystem. This generated model consists of one subsystem that contains blocks to program the FPGA and communicate with the FPGA I/O board during download and real-time execution of the target application. You add this generated subsystem to your xPC Target domain model.

For more on HDL Workflow Advisor, see .

Before you start HDL Workflow Advisor, develop a plan to map the FPGA subsystem inports and outports. Inports and outports may transmit signal data between the target computer and the FPGA over the PCI bus or map to I/O channels for communicating with external devices. In addition to the **Port Name** and **Port Type** (Inport or Outport), you need the following information to specify the I/O interface:

- **Data Type**—Encodes such attributes as width and sign. Data types must map consistently to their corresponding I/O pins. For instance, an inport of type `uint32` cannot be connected to an FPGA I/O interface of type `TTL I/O channel [0:7]` because too few TTL bits are available.
- **Target Platform Interfaces**—Encodes the I/O channels on the FPGA as well as their functional type. For a single-ended interface (TTL, LVCMOS), one channel maps to one connector pin. For a differential interface (RS422, LVDS), one channel maps to two connector pins. To discover the mapping for a particular pin, see the pin connector map provided with the board description. I/O channels may also map to a predefined specification or role (PCI Interface, Interrupt from FPGA).
- **Bit Range/Address/FPGA Pin**—Encodes the pins on the target platform to which the inports and outports are assigned, along with the channel number used by the port. For specification `PCI Interface`, encodes the PCI address used by the port.

In addition, if you specify vector inports or outports to a subsystem, HDL Workflow Advisor automatically inserts a strobe to eliminate race conditions between the elements. You specify a vector port as follows:

- **Inport** — Add a mux outside the subsystem connected to a demux inside the subsystem.

- **Outport** – Add a mux inside the subsystem connected to a demux outside the subsystem.
- **Inport and Outport** – Configure the port dimension to be greater than 1.

---

**Tip** Before generating code involving vector inports or outports, select the **Scalarize vector ports** check box in the Configuration Parameters dialog box, **HDL Code Generation > Global Settings > Coding style**.

---

For connector pin and I/O channel assignments of your supported FPGA I/O board, see “Speedgoat” in the *xPC Target I/O Reference*. For more on mapping Speedgoat FPGA I/O pins in HDL Workflow Advisor, see .

To start HDL Workflow Advisor, open your FPGA domain model. Try these steps with the `dxpcSGI0301servo_fpga` example:

- 1** In the FPGA model (`dxpcSGI0301servo_fpga`), right-click the FPGA subsystem (`ServoSystem`). From the context menu, select **HDL Code Generation > HDL Workflow Advisor**.

The **HDL Workflow Advisor** dialog window displays a number of tasks for the subsystem. You need to address only a subset of the tasks.

- 2** Expand the **Set Target** folder and select task **1.1 Set Target Device and Synthesis Tool**.
  - a** Set **Target Workflow** to **FPGA Turnkey**.
  - b** From the **Target platform** drop-down list, select the Speedgoat FPGA I/O board installed in your target computer.

For the `dxpcSGI0301servo_fpga.mdl` example, this is Speedgoat I0301 FPGA IO board (Acromag PMC-DX501).

- c** Click the **Run This Task** button.
- 3** In the same folder, select task **1.2 Set Target Interface**.

The **Set Target Interface** pane contains a table in which you specify the target platform interfaces. For each inport and outport signal, select the desired interface.

- a For signals between the target computer and the FPGA—In the **Target Platform Interfaces** column, select **PCI Interface**.
- b For signals through I/O lines (channels)—In the **Target Platform Interfaces** column, select the required I/O channel type (for example, **TTL I/O Channel [0:63]**).
- c In the **Bit Range/Address/FPGA Pin** column, enter the channel value for each signal.

---

**Tip** For **PCI Interface** signals, use the automatically generated values. Do not enter PCI address values.

---

- d After specifying interfaces for all signals, click **Run This Task**.
- 4 In the same folder, select task **1.3 Set Target Frequency** (optional).

The **Set Target Frequency** pane contains fields showing the FPGA input clock frequency (fixed) and the FPGA system clock frequency, which defaults to the FPGA input clock frequency.

- a To specify a different system clock frequency, type the new value into the field **FPGA system clock frequency (MHz)**.

---

**Note** If the specified value cannot be exactly generated, HDL Workflow Advisor will generate the closest match based on the following formula:

$$F_{system} = F_{input} * ClkFxMultiply / ClkFxDivide$$

where **ClkFxMultiply** and **ClkFxDivide** are integers.

---

- b Click **Run This Task**.
- 5 To complete the remaining tasks, expand the **Download to Target** folder, and right-click task **5.2 Generate xPC Target Interface**.
- 6 In this pane, click **Run To Selected Task**.



This action:

- Runs all remaining tasks.
- Creates the FPGA bitstream file in the `hdlsrc` folder. The xPC Target interface subsystem references this bitstream file during the build and download process.
- Generates a model named `gm_fpgamodelname_xpc.mdl`, which contains the xPC Target interface subsystem.

The following graphic shows the HDL Workflow Advisor folders for this process and the xPC Target interface subsystem contained in `gm_fpgamodelname_xpc`.

The image shows three overlapping windows from the HDL Workflow Advisor tool:

- Top Window (dxpcSGIO301servo\_fpga):** Displays a "Valid Motor Commands" list (0 to 180 deg, 360 (autorotate ON), 720 (autorotate OFF)) and a status bar indicating "FPGA board clock frequency: 33MHz". A "Motor Command" input is set to 360.
- Middle Window (HDL Workflow Advisor - dxpcSGIO301servo\_fpga/ServoSystem):** Shows a task list where "5.2. Generate xPC Target interface" is selected. The right pane shows the analysis results: "Generate xPC Target interface" passed, with a message: "Passed Generate xPC Target Interface. Generating new xPC Target interface model: gm\_dxpcSGIO301servo\_fpga\_xpc.mdl".
- Bottom Window (gm\_dxpcSGIO301servo\_fpga\_xpc \*):** Displays the generated xPC Target interface block diagram. It features a central "ServoSystem" block with inputs for Motor Command, LED1-4 Commands, and a HW Counter. The Motor Command is processed through a gain block (-K) and a double block. The LED commands are processed through gain blocks (-K) and double blocks. The outputs are connected to Scope1 and Scope2.

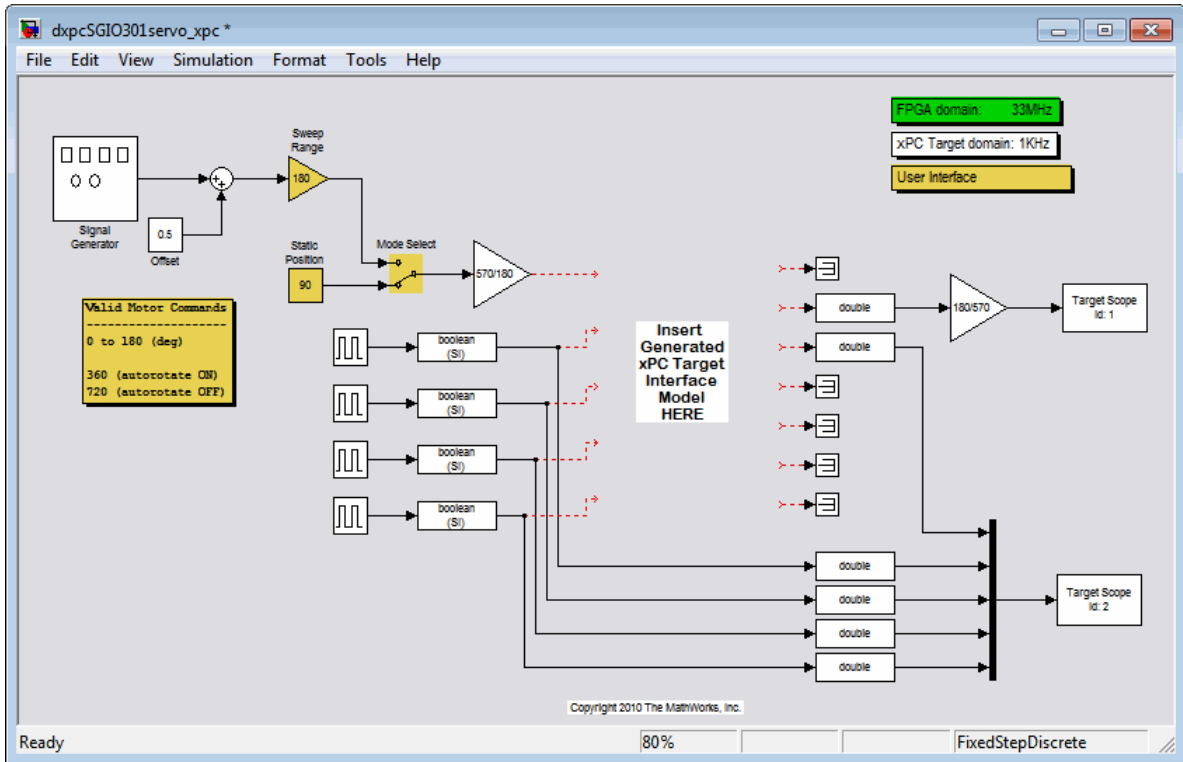
For information on how to integrate the interface subsystem into the xPC Target domain model, see “Adding the xPC Target Interface Subsystem to the xPC Target Domain Model” on page 1-14.

## Creating an xPC Target Domain Model

The xPC Target software enables you to execute Simulink and Stateflow<sup>®</sup> models on a target computer for rapid control prototyping, hardware-in-the-loop (HIL) simulation, and other real-time testing applications. You can also include Speedgoat FPGA I/O boards in your design. Either before or after you have created the FPGA domain model and the xPC Target interface subsystem using HDL Workflow Advisor, create an xPC Target domain model in which you plan to include the interface subsystem.

- 1** Create a Simulink model that contains the functionality you want to simulate in conjunction with the FPGA algorithm. This model, referred to as the xPC Target domain model, runs in real-time on the target computer. The xPC Target model and the FPGA algorithm communicate over the PCI bus.
- 2** Save the model.

See `dxpcSGI0301servo_xpc` for an example of an xPC Target domain model. In this example, the disconnected signals later connect to the inports and outports of the xPC Target interface subsystem when you add it. See “Adding the xPC Target Interface Subsystem to the xPC Target Domain Model” on page 1-14.



## Adding the xPC Target Interface Subsystem to the xPC Target Domain Model

This topic assumes that you have generated an xPC Target interface subsystem with the HDL Coder software. If you have not yet done so, see “Generating HDL with the Workflow Advisor” on page 1-7.

- 1 Open `gm_fpgamodelname_xpc.mdl` in the Simulink editor.

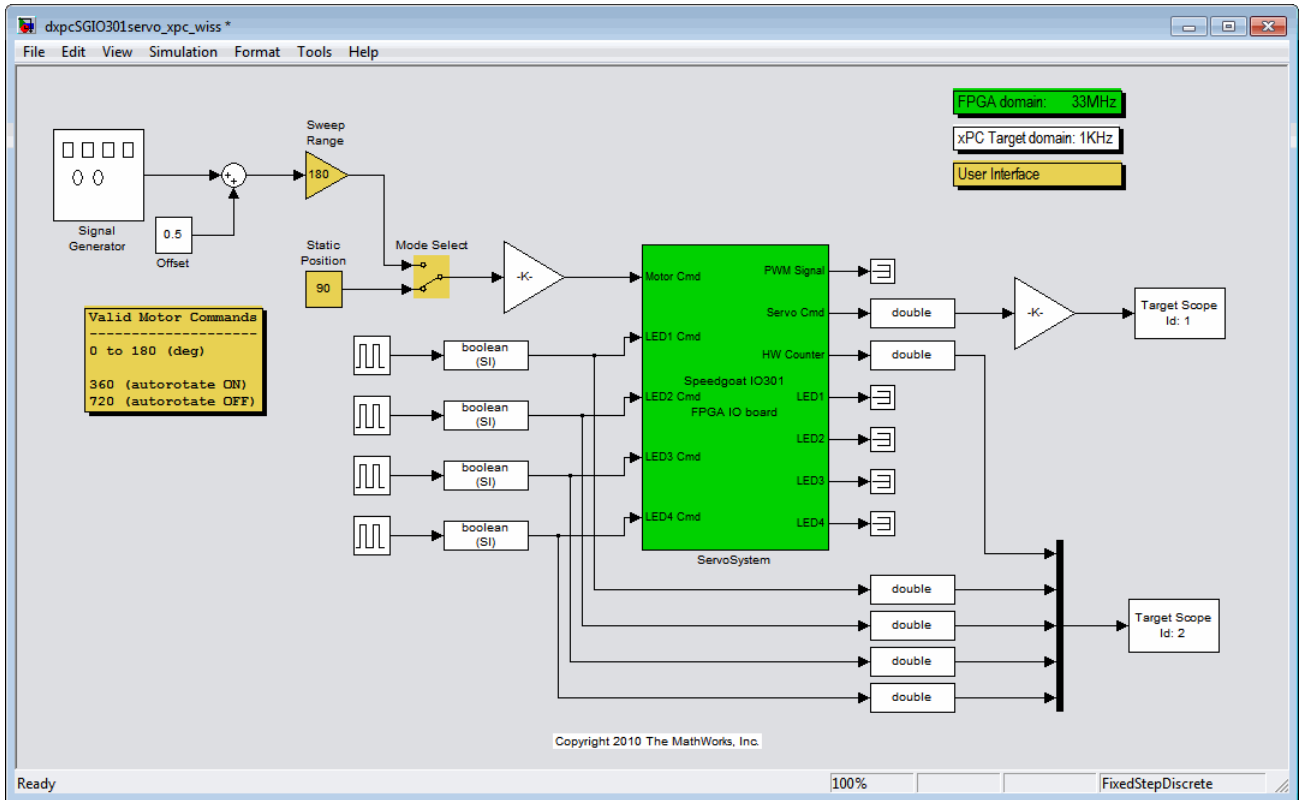
This generated model contains a subsystem with the same name as the subsystem in the Simulink FPGA domain model. Although the appearance is similar, this subsystem does not contain the Simulink algorithm. Instead, the algorithm is implemented in an FPGA bitstream. You reference and load this algorithm onto the FPGA from this subsystem.

- 2** Select, copy, and paste the this subsystem, xPC Target interface subsystem, into the xPC Target domain model.
- 3** Save or discard `gm_fpgamodelname_xpc.mdl`. You can always recreate it using the HDL Workflow Advisor.
- 4** In the xPC Target domain model, connect signals to the inports and outports of the xPC Target interface subsystem.

The xPC Target interface subsystem is a masked subsystem with three parameters:

- Device index
  - PCI slot
  - Sample time
- 5** Set the parameters according to the FPGA I/O boards in your target computer.
    - a** If you have a single FPGA I/O board, leave the device index and PCI slot at the default values. You can set the sample time or leave it at `-1` for inheritance.
    - b** If you have multiple FPGA I/O boards, give each board a unique device index.
    - c** If you have two or more boards of the same type (for example, two IO301s), specify the PCI slot ([bus, slot]) for each board. Get this information with the `xpctarget.xpc.getxpcpci` function.
  - 6** Save the model.

See `dxpcSGI0301servo_xpc_wiss` for an example of an xPC Target domain model that has the interface subsystem pasted and connected.



You are now ready to build and download the xPC Target domain model. See “Building and Downloading the Target Application” on page 1-16.

## Building and Downloading the Target Application

This topic assumes that you have created an xPC Target domain model that includes an xPC Target interface subsystem generated from the HDL Workflow Advisor. If you have not yet done so, see “Adding the xPC Target Interface Subsystem to the xPC Target Domain Model” on page 1-14.

- 1 Configure the target computer and connect it to the host computer.

- 2** Build and download the xPC Target model. The xPC Target model loads onto the target computer and the FPGA algorithm bitstream loads onto the FPGA.
- 3** If you are using I/O lines (channels), confirm that you have connected the lines to your external hardware under test.

The start and stop of the xPC Target model controls the start and stop of the FPGA algorithm. The FPGA algorithm executes at the clock frequency of the FPGA I/O board, while the application executes in accordance with the model sample time.

## Interrupts


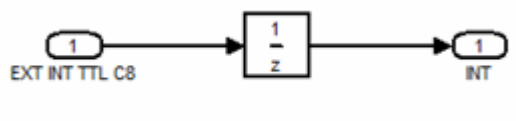
xPC Target software schedules the target application using either the internal timer of the target computer (default) or an interrupt from an I/O board. You can use your Speedgoat FPGA board to generate an interrupt, which allows you to:

- Schedule execution of the target application based on this interrupt (synchronous execution). This method assumes you generate the interrupt periodically.
- Execute a designated subsystem in your target application (asynchronous execution).

To use FPGA-based interrupts, set up and configure the FPGA domain and xPC Target domain models.

## FPGA Domain Model

In the FPGA domain subsystem, create the interrupt source execution of the target application in one of the following ways:

Source	Description
Internal	<p>A clock you create using Simulink blocks to create input signals. This clock is a binary pulse train of zeros and ones (transition from 0 to 1 and vice versa). The clock generates an interrupt on a rising edge. The following is an example of an internally generated interrupt source from Simulink blocks. Connect the internally generated interrupt source to an output labelled INT.</p>  <p>The diagram shows a rectangular block labeled 'Interrupt Source' with 'Int Timer' and '1ms' below it. An arrow labeled 'ufix1' points from the block to an output port labeled 'INT' with a '5' inside a circle.</p>
External	<p>A clock signal that comes from a device outside the target computer. You use a digital input pin to connect to this signal. The following is an example of an externally generated interrupt source that comes from TTL channel 8. Delay this source by one FPGA clock cycle and connect to an output labeled INT.</p>  <p>The diagram shows an input port labeled 'EXT INT TTL C8' with a '1' inside a circle. An arrow points to a square block containing '1', '-', and 'z'. Another arrow points from this block to an output port labeled 'INT' with a '1' inside a circle.</p>

In both cases, wire the interrupt source to an output in the FPGA subsystem and assign the output as `Interrupt` from `FPGA` in the HDL Workflow Advisor task 1.2 **Set Target Interface**.

You are now ready to set up interrupt support in the xPC Target domain model. See “xPC Target Domain Model” on page 1-18.

### xPC Target Domain Model

If you have not yet done so, see “FPGA Domain Model” on page 1-17.

Configure the model xPC Target domain model to set up interrupt support:



- 1** Open the xPC Target domain model.
- 2** In the Simulink editor, select **Simulation > Configuration Parameters**.
- 3** Navigate to **Code Generation > xPC Target options**.
- 4** From the **Real-time interrupt source** list, select one of the following:
  - Auto (PCI only).
  - The IRQ assigned to your FPGA board.
- 5** From the **I/O board generating the interrupt** parameter, select your FPGA board, for example, Speedgoat\_I0301.
- 6** Add the xPC Target interface subsystem to the model (see “Adding the xPC Target Interface Subsystem to the xPC Target Domain Model” on page 1-14).
- 7** Build and download the application to the target computer.
- 8** When you start the target application, simulation updates occur when the application receives an interrupt from the FPGA I/O board.

## Demos

The xPC Target product provides these demos as examples of FPGA applications using the integrated HDL Coder and xPC Target workflow:

<b>Demo</b>	<b>Description</b>
Servo Control with the Speedgoat IO301 FPGA Board	Demonstrates programming and configuring the Speedgoat IO301 with a simple PWM servo controller, hardware counter, and digital I/O.
Digital I/O with the Speedgoat IO303 FPGA Board	Demonstrates programming and configuring the Speedgoat IO303 for digital I/O.

# Vector CANape Support

---

This topic describes how to use xPC Target to interface the target computer to the Vector CAN Application Environment (CANape) (<http://www.vector-worldwide.com>) using the Universal Calibration Protocol (XCP). This chapter includes the following sections:

- “Vector CANape” on page 2-2
- “Configuring the xPC Target and Vector CANape Software” on page 2-4
- “Event Mode Data Acquisition” on page 2-11

# Vector CANape

In this section...
“Introduction” on page 2-2
“xPC Target and Vector CANape Limitations” on page 2-3

## Introduction

You can use a target computer as an electronic control unit (ECU) for a Vector CANape® system. Using a target computer in this way, a Vector CANape system can read signals and parameters from a target application running on the target computer.

The xPC Target software supports polling and event driven modes for data acquisition. Polling mode data acquisition is straightforward. Event mode data acquisition requires additional settings (see “Event Mode Data Acquisition” on page 2-11).

---

**Note** This chapter describes how to configure xPC Target and Vector CANape software to work together. It also assumes that you are familiar with the Vector CANape product family. See <http://www.vector-cantech.com> for further information about the Vector CANape products.

---

The xPC Target software works with Vector CANape version 5.6 and higher. To enable a target computer to work with Vector CANape software, you need to:

- Configure Vector CANape to communicate with the xPC Target software as an ECU.
- Enable the xPC Target software to generate a target application that can provide data compliant with Vector CANape.
- Provide a standard TCP/IP physical layer between the host computer and target computer. The xPC Target software supports Vector CANape only through TCP/IP.

To support the XCP communication layer, the xPC Target software provides:

- An XCP server process in the target application that runs on-demand in the background.
- A generator that produces A2L (ASAP2) files that Vector CANape can load into the Vector CANape software database. The generated file contains signal and parameter access information for the target application.

### **xPC Target and Vector CANape Limitations**

The xPC Target software supports the ability to acquire signal data at the base sample rate of the model. The xPC Target software does not support the following for Vector CANape:

- Vector CANape start and stop ECU (target computer) commands

---

**Tip** To start and stop the application on the target computer, use the xPC Target start and stop commands, for example `tg.start`, `tg.stop`.

---

- Vector CANape calibration commands or flash RAM calibration commands
- Multiple simultaneous Vector CANape connections to a single target computer

## Configuring the xPC Target and Vector CANape Software

### In this section...

“Setting Up and Building the Model” on page 2-4

“Creating a New Vector CANape Project to Associate with a Particular Target Application” on page 2-6

“Configuring the Vector CANape Device” on page 2-7

“Providing A2L (ASAP2) Files for the Vector CANape Database” on page 2-10

### Setting Up and Building the Model

Set up your model to work with Vector CANape. This procedure uses the `xpcosc` model. It assumes that you have already configured your model to generate xPC Target code. If you have not done so, see “Entering Simulation Parameters” and “xPC Target Options Configuration Parameter” on page 4-2 in the xPC Target™ User’s Guide on page 1. It also assumes that you have already created a Vector CANape project. If you have not done so, see “Creating a New Vector CANape Project to Associate with a Particular Target Application” on page 2-6.

**1** In the MATLAB® Command Window, type

```
xpcosc
```

**2** Open the xPC Target library. For example, in the MATLAB window, type

```
xpclib
```

**3** Navigate to the Misc sublibrary and double-click that library.

**4** Drag the XCP Server block to the `xpcosc` model.

This block enables an XCP server process to run in the target application.

**5** In the model, double-click the XCP Server block. Check the following parameters:

- **Target Address** — Target IP address for target computer. The default value is `getxpcenv('TcpIpTargetAddress')`. Typically, you will want to leave the default entry. Otherwise, enter the TCP/IP address for the target computer.
  - **Server Port** — Port for communication between target computer and XCP server. The default value is 5555. This value must be the same as the port number you specify for the Vector MATLAB device.
- 6 If you want to use the event mode to acquire signal data, set the priority of the `xpcserver` block to be the lowest priority. For example, enter a priority of 10000000. For Simulink blocks, the higher the priority number, the lower the priority.
  - 7 In the model Simulink window, click **Simulation > Configuration Parameters**.

The Configuration Parameters dialog box is displayed for the model.

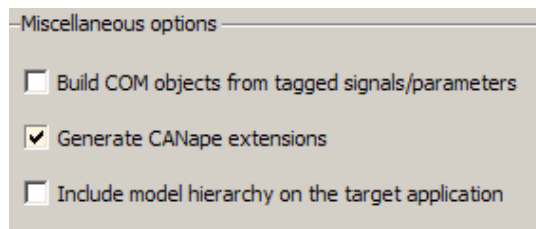
- 8 In the left pane, click the **xPC Target options** node.

The associated pane is displayed.

- 9 In the **Miscellaneous options** area, select the **Generate CANape extensions** check box.

This option enables target applications to generate data, such as that for A2L (ASAP2), for Vector CANape.

The **Miscellaneous options** are of the **xPC Target options** node should now look like the figure shown.



- 10 Build the model.

The xPC Target software builds the target application, including an A2L (ASAP2) data file for the target application.

- 11 On the target computer monitor, look for the following message. These messages indicate that you have built the target application without producing an error and can now connect to the target with Vector CANape.

```
XCP Server set up, waiting for connection
```

You can now create a new Vector CANape project (see “Creating a New Vector CANape Project to Associate with a Particular Target Application” on page 2-6).

### **Creating a New Vector CANape Project to Associate with a Particular Target Application**

This procedure describes how to create a new Vector CANape project that can communicate with an xPC Target application. It assumes that you have set up, built, and downloaded your model (see “Setting Up and Building the Model” on page 2-4).

- 1 In a DOS window, create a new directory to hold your project. This can be the same directory as your xPC Target model files. For example, type

```
mkdir C:\MyProject
```

- 2 Start Vector CANape.
- 3 Select **File > New project**.

A new project wizard is displayed. Follow this dialog to create a new project.

- 4 After you create the new project, start it.

After the preliminary warning, the CANape window is displayed.

You can now configure the target computer and the loaded target application as a Vector CANape device (see “Configuring the xPC Target and Vector CANape Software” on page 2-4).



## Configuring the Vector CANape Device

This procedure describes how to configure the Vector CANape Device to work with your target application. It assumes the following:

- You have created a new Vector CANape project to associate with a particular target application. If you have not yet done so, see “Creating a New Vector CANape Project to Associate with a Particular Target Application” on page 2-6.
- You have set up, built, and downloaded your model. If you have not yet done so, see “Setting Up and Building the Model” on page 2-4.

**1** If you have not yet started your new Vector CANape project, start it now.

The Vector CANape window is displayed.

**2** In the CANape window, click **Device > Device configuration**.

The device configuration window is displayed.

**3** In the device configuration window, click **New**.

**4** In **Device Name**, enter a name for the device to describe your target application. For example, type

xPCTarget

Add any required comments.

**5** Click **Next**.

**6** From the driver-type menu list, select **XCP**.

**7** Click **Driver settings**.

The XCP driver settings window is displayed.

**8** In the **Transport layer** pane, from the **Interface** menu list, select TCP.

**9** In the **Transport layer** pane, click **Configuration**.

**10** In the **Host** field, enter the IP address of your target computer.

This is the target computer to which you have downloaded the target application.

**11** Set the port number to 5555.

**12** Click **OK**.

**13** If you have Vector CANape Version 5.6.32.3 and higher, and you want to use the xPC Target software to acquire event driven data:

- a** In the **Driver** pane of the XCP driver settings window, click **Extended driver settings**.
- b** Set the **ODT\_ENTRY\_ADDRESS\_OPT\_DISABLED** parameter to Yes.

With this setting, events that are generated in the xPC Target environment will be based on the model base sample time. For example, a sample time of 0.001 seconds will appear as 100 milliseconds.

**c** Click **OK**.

**14** In the XCP driver settings window, verify the connection to the target computer by clicking **Test connection**. This command succeeds only if the target computer is running and connected to the host computer. Be sure that no other host is connected to the target computer.

**15** Click **OK**.

The **Device** dialog is displayed.

**16** Click **Next**.

Do not exit the dialog.

You can now configure the location of the target application A2L (ASAP2) file for the CANape database. See “Configuring the Location of the A2L (ASAP2) File” on page 2-9.

If you want to load a new target application, you must close Vector CANape, download a new target application through the MATLAB interface, then restart Vector CANape.

## Configuring the Location of the A2L (ASAP2) File

Use this procedure to configure the location of the target application A2L (ASAP2) file for Vector CANape. This procedure assumes that you have already configured the Vector CANape device and are still in the device configuration dialog.

**1** Clear **Automatic detection of the database name**.

**2** At the **Database name** parameter, click **Browse**.

The **Select database for device xPCTarget** dialog is displayed.

**3** Browse to the directory that contains the A2L (ASAP2) file for the target application.

This might be the directory in which you built the target application, or it might be the directory you specified during the target application build configuration.

**4** Select the A2L (ASAP2) file. Click **Open**.

A dialog requests confirmation of ASAP2 settings.

**5** Click **Yes**.

**6** Click **Next**.

**7** Click **Next**.

**8** Click **Next**.

**9** Click **OK**.

**10** You have completed the configuration of Vector CANape for the xPC Target software environment.

You can now monitor and control your xPC Target system. The CANape database should be populated with a comprehensive list of target application signals and parameters that are available. See “Event Mode Data Acquisition” on page 2-11.

During target application changes, you might need to manually reload the A2L (ASAP2) that is generated by the xPC Target build process. You can do this from the CANape Database editor.

### **Providing A2L (ASAP2) Files for the Vector CANape Database**

This topic assumes that:

- You have set up and built your model to generate data for Vector CANape. If you have not yet done so, see “Setting Up and Building the Model” on page 2-4.
- You have created a Vector CANape project directory and know the name of that project directory.

To enable Vector CANape to load the A2L (ASAP2) file for the model `xpcosc`:

- 1** In a DOS window, change directory to the one that contains the A2L (ASAP2) file from the previous procedure. For example:

```
cd D:\work\xpc
```

- 2** Look for and copy the A2L (ASAP2) file to your Vector CANape project directory. For example:

```
copy xpcosc.a2l C:\MyProject
```

Vector CANape automatically loads the target application A2L (ASAP2) file when it connects to the target computer.

# Event Mode Data Acquisition

In this section...
“Guidelines” on page 2-11
“Limitations” on page 2-11

## Guidelines

To acquire event mode data rather than polling data, note the following guidelines:

- Set the priority of the xpcserver block to the lowest possible. See suggested priority values in “Setting Up and Building the Model” on page 2-4.
- The xPC Target software generates events at the base sample rate; this execution rate is the fastest possible. If you are tracing a signal that is updated at a slower rate than the base sample rate, you must decimate the data to match the actual execution. (The xPC Target software generates the event name with the ASAP2 generation during model code generation.)
- You can associate signals with the event generation through the Vector CANape graphical user interface.

See the Vector CANape documentation for further details on associating events with signals.

## Limitations

The event mode data acquisition has the following limitations:

- Every piece of data that the xPC Target software adds to the event list slows down the target application. The amount of data that you can observe depends on the model sample time and the speed of the target computer. It is possible to overload the target computer CPU to the point where data integrity is reduced.
- You can only trace signals and scalar parameters. You cannot trace vector parameters.



# Incorporating Fortran S-Functions

---

- “Fortran S-Functions” on page 3-2
- “Atmosphere Model Example” on page 3-5

## Fortran S-Functions

The xPC Target product supports Fortran in Simulink models using S-functions. For more details, see “Creating Fortran S-Functions” in the *Developing S-Functions* guide, particularly the sections “Creating Level-2 Fortran S-Functions” and “Porting Legacy Code”.

In this section...
“Prerequisites” on page 3-2
“Simulink Demos Folder” on page 3-2
“Steps to Incorporate Fortran” on page 3-3

### Prerequisites

You must have xPC Target Version 1.3 or later to use Fortran for xPC Target applications. The xPC Target product supports the Fortran compiler(s) listed here:

[http://www.mathworks.com/support/compilers/current\\_release/](http://www.mathworks.com/support/compilers/current_release/)

### Simulink Demos Folder

The Simulink demos folder contains a tutorial and description on how to incorporate Fortran code into a Simulink model using S-functions. To access the tutorial and description,

- 1 In the MATLAB Command Window, type

```
demods
```

A list of MATLAB products appears on the left side of the MATLAB Online Help window.

- 2 From the left side of the window, select **Simulink > Demos > Modeling Features**.

A list of Simulink examples appears.



**3 Click Custom Code and Hand Coded Blocks using the S-function API.**

The associated Simulink demos page opens.

**4 Click Open this model.**

S-function examples are displayed.

**5 Double-click the Fortran S-functions block.**

Fortran S-functions and associated templates appear.

## Steps to Incorporate Fortran

This topic lists the general steps to incorporate Fortran code into an xPC Target application. Detailed commands follow in the accompanying examples.

- 1** Using the Fortran compiler, compile the Fortran code (subroutines (\*.f)). You will need to specify particular compiler options.
- 2** Write a Simulink C-MEX wrapper S-function. This wrapper S-function calls one or more of the Fortran subroutines in the compiled Fortran object code from step 1.
- 3** Use the `mex` function to compile this C-MEX S-function using a Visual C/C++ compiler. Define several Fortran run-time libraries to be linked in.

This step creates the Simulink S-function MEX-file.

- 4** Run a simulation C-MEX file with the Simulink software to validate the compiled Fortran code and wrapper S-function.
- 5** Copy relevant Fortran run-time libraries to the application build folder for the xPC Target application build.
- 6** Define the Fortran libraries, and the Fortran object files from step 1, in the Simulink Coder™ dialog box of the Simulink model. You must define these libraries and files as additional components to be linked in when the xPC Target application link stage takes place.

- 7 Initiate the xPC Target specific Simulink Coder build procedure for the demo model. Simulink Coder builds and downloads xPC Target onto the target PC.

# Atmosphere Model Example

## In this section...

“In This Example” on page 3-5

“Creating a Fortran Atmosphere Model” on page 3-5

“Compiling Fortran Files” on page 3-7

“Creating a C-MEX Wrapper S-Function” on page 3-8

“Compiling and Linking the Wrapper S-Function” on page 3-12

“Validating the Fortran Code and Wrapper S-Function” on page 3-14

“Preparing the Model for the xPC Target Application Build” on page 3-14

“Building and Running the xPC Target Application” on page 3-16

## In This Example

This example uses the demo Atmosphere model that comes with the Simulink product. The following procedures require you to know how to write Fortran code according to Simulink and xPC Target software requirements. See “Creating Fortran S-Functions” in the *Developing S-Functions* guide for these details.

Before you start, create an xPC Target Simulink model for the Atmosphere model. See “Creating a Fortran Atmosphere Model” on page 3-5.

## Creating a Fortran Atmosphere Model

To create an xPC Target Atmosphere model in Fortran, you need to add an xPC Target Scope block to the `sfcn_demo_atmos` model. Perform this procedure if you do not already have an xPC Target Atmosphere model for Fortran.

- 1 From the MATLAB window, change folder to the working folder, for example, `xpc_fortran_test`.

#### 2 Type

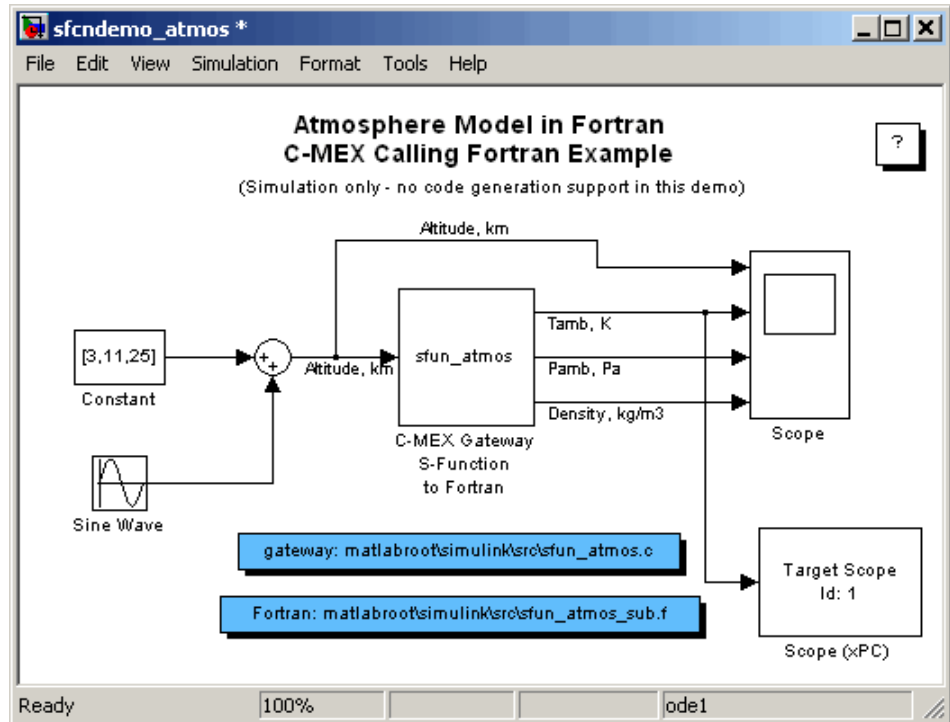
`sfcndemo_atmos`

The `sfcndemo_atmos` model is displayed.

#### 3 Add an xPC Target Scope block of type Target.

#### 4 Connect this Scope block to the `Tamb, K` signal.

The model `sfcndemo_atmos.mdl` should look like the figure shown.



#### 5 Double-click the target Scope block.

#### 6 From the **Scope mode** parameter, choose Graphical rolling.

#### 7 For the **Number of samples** parameter, enter 240.

- 8** Click **Apply**, then **OK**.
- 9** Double-click the Sine Wave block.
- 10** For the **Sample time** parameter, enter 0.05.
- 11** Click **OK**.
- 12** From the **File** menu, click **Save as**. Browse to your current working folder, for example, `xpc_fortran_test`. Enter a filename. For example, enter `fortran_atmos_xpc` and then click **Save**.

Your next task is to compile Fortran code. See “Compiling Fortran Files” on page 3-7.

## Compiling Fortran Files

- 1** In the MATLAB Command Window, copy the file `sfun_atmos_sub.F` into your Fortran working folder, for example, `xpc_fortran_test`. This is the sample Fortran code that implements a subroutine for the Atmosphere model.
- 2** From `Fortran_compiler_dir\lib\ia32`, copy the following files to the working folder:
  - `libifcore.lib`
  - `libifcoremd.lib`
  - `ifconsol.lib`
  - `libifportmd.lib`
  - `libifport.lib`
  - `libmmd.lib`
  - `libm.lib`
  - `libirc.lib`
  - `libmmt.lib`
  - `libifcoremt.lib`

- `svml_disp.lib`

**3** From a DOS prompt, change folder to the working folder and create the object file. For example:

```
ifort /fpp /Qprec /c /nologo /MT /fixed /iface:cref -Ox sfun_atmos_sub.F
```

Your next task is to create a wrapper S-function. See “Creating a C-MEX Wrapper S-Function” on page 3-8.

## Creating a C-MEX Wrapper S-Function

This topic describes how to create a C-MEX wrapper S-function for the Fortran code in `sfun_atmos_sub.f`. This function is a level 2 S-function. It incorporates existing Fortran code into a Simulink S-function block and lets you execute Fortran code from the Simulink software. Before you start,

- Compile your Fortran code. See “Compiling Fortran Files” on page 3-7.
- Be familiar with writing Simulink S-functions. In particular, read “Creating Fortran S-Functions” in the *Developing S-Functions* guide. Note the “Creating Level-2 Fortran S-Functions” section. This section lists guidelines for creating Fortran level 2 S-functions, including calling conventions.
- Refer to “S-Function Callback Methods — Alphabetical List” in the *Developing S-Functions* guide. The Simulink software invokes these methods when simulating a model with S-functions.
- Refer to the “S-Function SimStruct Functions — Alphabetical List” in the *Developing S-Functions* guide for detailed descriptions of the functions that access the fields of an S-function’s simulation data structure (`SimStruct`). S-function callback methods use these functions to store and retrieve information about an S-function.

The following procedure outlines the steps to create a C-MEX wrapper S-function to work with `sfun_atmos_sub.f`. It uses the template file `sfuntmpl_gate_fortran.c`.

---

**Note** This topic describes how to create a level 2 Fortran S-function for the `fortran_atmos_xpc` model. This file is also provided in `sfun_atmos.c`.

---

- 1 Copy the file `sfunmpl_gate_fortran.c` to your working folder.

This is your C-MEX file for calling into your Fortran subroutine. It works with a simple Fortran subroutine.

- 2 With a text editor of your choice, open `sfunmpl_gate_fortran.c`.

- 3 Inspect the file. This is a self-documenting file.

This file contains placeholders for standard Fortran level 2 S-functions, such as the S-function name specification and Simulink callback methods.

- 4 In the `#define S_FUNCTION_NAME` definition, add the name of your S-function. For example, edit the definition line to look like

```
#define S_FUNCTION_NAME sfun_atmos
```

- 5 In the file, read the commented documentation for fixed-step and variable-step fixed algorithm support.

- 6 Delete or comment out the code for fixed-step and variable-step fixed-algorithm support. You do not need these definitions for this example.

- 7 Find the line that begins `extern void nameofsub_`. Specify the function prototype for the Fortran subroutine. For the `sfun_atmos_sub.obj` executable, the Fortran subroutine is `atmos_`. Replace

```
extern void nameofsub_(float *sampleArgs, float *sampleOutput);
```

with

```
extern void atmos_(float *falt, float *fsigma, float *fdelta, float *ftheta);
```

Enter a `#if defined/#endif` statement like the following for Windows compilers.

```
#ifdef _WIN32
#define atmos_ atmos
```

```
#endif
```

- 8** Add a typedef to specify the parameters for the block. For example,

```
typedef enum {TO_IDX=0, PO_IDX, RO_IDX, NUM_SPARAMS } paramIndices;
```

```
#define TO(S) (ssGetSFcnParam(S, TO_IDX))
```

```
#define PO(S) (ssGetSFcnParam(S, PO_IDX))
```

```
#define RO(S) (ssGetSFcnParam(S, RO_IDX))
```

- 9** Use the `mdlInitializeSizes` callback to specify the number of inputs, outputs, states, parameters, and other characteristics of the S-function. S-function callback methods use `SimStruct` functions to store and retrieve information about an S-function. Be sure to specify the temperature, pressure, and density parameters. For example,

```
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S, NUM_SPARAMS); /* expected number */
    #if defined(MATLAB_MEX_FILE)
        if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) goto EXIT_POINT;
    #endif

    {
        int iParam = 0;
        int nParam = ssGetNumSFcnParams(S);

        for ( iParam = 0; iParam < nParam; iParam++ )
        {
            ssSetSFcnParamTunable( S, iParam, SS_PRM_SIM_ONLY_TUNABLE );
        }
    }

    ssSetNumContStates( S, 0 );
    ssSetNumDiscStates( S, 0 );
    ssSetNumInputPorts(S, 1);
    ssSetInputPortWidth(S, 0, 3);
    ssSetInputPortDirectFeedThrough(S, 0, 1);
    ssSetInputPortRequiredContiguous(S, 0, 1);
    ssSetNumOutputPorts(S, 3);
    ssSetOutputPortWidth(S, 0, 3); /* temperature */
}
```



```

    ssSetOutputPortWidth(S, 1, 3); /* pressure */
    ssSetOutputPortWidth(S, 2, 3); /* density */

#if defined(MATLAB_MEX_FILE)
EXIT_POINT:
#endif
    return;
}

```

- 10** Use the `mdlInitializeSampleTimes` callback to specify the sample rates at which this S-function operates.

```

static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
    ssSetModelReferenceSampleTimeDefaultInheritance(S);
}

```

- 11** Use the `mdlOutputs` callback to compute the signals that this block emits.

```

static void mdlOutputs(SimStruct *S, int_T tid)
{
    double *alt = (double *) ssGetInputPortSignal(S,0);
    double *T   = (double *) ssGetOutputPortRealSignal(S,0);
    double *P   = (double *) ssGetOutputPortRealSignal(S,1);
    double *rho = (double *) ssGetOutputPortRealSignal(S,2);
    int     w   = ssGetInputPortWidth(S,0);
    int     k;
    float   falt, fsigma, fdelta, ftheta;

    for (k=0; k<w; k++) {

        /* set the input value */
        falt = (float) alt[k];

        /* call the Fortran routine using pass-by-reference */
        atmos_(&falt, &fsigma, &fdelta, &ftheta);

        /* format the outputs using the reference parameters */
        T[k] = mxGetScalar(T0(S)) * (double) ftheta;
    }
}

```

```
        P[k]   = mxGetScalar(P0(S)) * (double) fdelta;  
        rho[k] = mxGetScalar(R0(S)) * (double) fsigma;  
    }  
}
```

- 12** Use the `mdlTerminate` callback to perform any actions required at termination of the simulation. Even if you do not have any operations here, you must include a stub for this callback.

```
static void mdlTerminate(SimStruct *S)  
{  
}
```

- 13** In the file, read the commented documentation for the following callbacks:
- `mdlInitializeConditions` — Initializes the state vectors of this S-function.
  - `mdlStart` — Initializes the state vectors of this S-function. This function is called once at the start of the model execution.
  - `mdlUpdate` — Updates the states of a block.

These are optional callbacks that you can define for later projects. You do not need to specify these callbacks for this example.

- 14** Delete or comment out the code for these callbacks.
- 15** Save the file under another name. For example, save this file as `sfun_atmos.c`. Do not overwrite the template file.
- 16** Copy the file `sfun_atmos.c` into your Fortran working folder, for example, `xpc_fortran_test`.

Your next task is to compile and link the wrapper S-function. See “Compiling and Linking the Wrapper S-Function” on page 3-12.

## Compiling and Linking the Wrapper S-Function

This topic describes how to create (compile and link) a C-MEX S-function from the `sfun_atmos.c` file. Before you start, copy the following files into the working folder, `xpc_fortran_test`. (You should have copied these files when you performed the steps in “Compiling Fortran Files” on page 3-7.)

- `libifcore.lib`
- `libifcoremd.lib`
- `ifconsol.lib`
- `libifportmd.lib`
- `libifport.lib`
- `libmmd.lib`
- `libm.lib`
- `libirc.lib`
- `libmmt.lib`
- `libifcoremt.lib`
- `svml_disp.lib`

Use the `mex` command with a C/C++ compiler such as Microsoft Visual C/C++ Version 6.0.

This topic assumes that you have created a C-MEX wrapper S-function. See “Creating a C-MEX Wrapper S-Function” on page 3-8.

Invoking the `mex` command requires you to compile the wrapper C file `sfun_atmos.c`. Be sure to link in the following:

- Compiled Fortran code: `sfun_atmos_sub.obj`
- Fortran run-time libraries to resolve external function references and provide the Fortran run-time environment

When you are ready, `mex` the code. For example

```
mex -v LINKFLAGS="$LINKFLAGS /NODEFAULTLIB:libcmt.lib libifcoremd.lib  
ifconsol.lib libifportmd.lib libmmd.lib libirc.lib svml_disp.lib" sfun_atmos.c  
sfun_atmos_sub.obj
```

---

**Note** The command and all its parameters must be on one line.

---

This command compiles and links the `sfun_atmos_sub.c` file. It creates the `sfun_atmos.mex` file in the same folder.

Your next task is to validate the Fortran code and wrapper S-function. See “Validating the Fortran Code and Wrapper S-Function” on page 3-14.

### Validating the Fortran Code and Wrapper S-Function

Validate the generated C-MEX S-function, `sfun_atmos.mex`. Bind the C-MEX S-function to an S-function block found in the Simulink block library. You can mask the S-function block like any other S-function block to give it a specific dialog box.

This topic assumes that you have compiled and linked a wrapper S-function. See “Compiling and Linking the Wrapper S-Function” on page 3-12.

The Atmosphere model example has a Simulink model associated with it.

- 1 In the MATLAB window, type

```
fortran_atmos_xpc
```

This opens the Simulink model associated with the Atmosphere model. This model includes an S-function block bound to `sfun_atmos.mex`.

- 2 Select the **Simulation** menu **Start** option to simulate the model.
- 3 Examine the behavior of the Atmosphere model by looking at the signals traced by the Scope block.

Your next task is to prepare the model to build an xPC Target application. See “Preparing the Model for the xPC Target Application Build” on page 3-14.

### Preparing the Model for the xPC Target Application Build

Before you build the Atmosphere model for xPC Target, define the following build dependencies:

- The build procedure has access to `sfun_atmos.sub.obj` for the link stage.

- The build procedure has access to the Fortran run-time libraries (see “Compiling and Linking the Wrapper S-Function” on page 3-12) for the link stage.

This topic assumes that you have validated the Fortran code and wrapper S-function (see “Validating the Fortran Code and Wrapper S-Function” on page 3-14).

- 1** In the MATLAB window, type

```
fortran_atmos_xpc
```

This opens the Simulink model associated with the Atmosphere model.

- 2** In the Simulink model, from the **Simulation** menu, click **Configuration Parameters**.

The Configuration Parameters dialog box appears.

- 3** In the left pane, click the **Code Generation** node.

The code generation pane opens.

- 4** In the **Target selection** section, click the **Browse** button at the **System target file** list.

- 5** Click `xpctarget.tlc`.

- 6** In the **Make command** field, replace `make_rtw` with one for the Fortran compiler.

```
make_rtw S_FUNCTIONS_LIB="..\sfun_atmos_sub.obj ..\libifcoremt.lib ..\libmmt.lib  
..\ifconsol.lib ..\libifport.lib ..\libirc.lib ..\svml_disp.lib"
```

---

**Note** The command and all its parameters must be on one line.

---

- 7** Click **Apply**.

- 8** Click **OK**.

9 From the **File** menu, click **Save**.

This command requires that the application build folder be the current folder (one level below the working folder, `xpc_fortran_test`). Because of this, all additional dependency designations must start with `.. \`.

Specify all Fortran object files if your model (S-Function blocks) depends on more than one file. For this example, you specify the run-time libraries only once.

Your next task is to build and run the xPC Target application. See “Building and Running the xPC Target Application” on page 3-16.

### **Building and Running the xPC Target Application**

This topic assumes that you have prepared the model to build an xPC Target application. See “Preparing the Model for the xPC Target Application Build” on page 3-14.

Build and run the xPC Target application as usual. Be sure that you have defined Microsoft Visual C/C++ as the xPC Target C compiler using.

After the build procedure succeeds, xPC Target automatically downloads the application to the target PC. The Atmosphere model already contains an xPC Target Scope block. This allows you to verify the behavior of the model. You will be able to compare the signals displayed on the target screen with the signals obtained earlier by the Simulink simulation run (see “Validating the Fortran Code and Wrapper S-Function” on page 3-14).

# Target Application Environment

---

- “xPC Target Options Configuration Parameter” on page 4-2
- “xPC Target Explorer” on page 4-3
- “Target Environment Command-Line Interface” on page 4-10
- “Setting Up the Target Application Environment” on page 4-14
- “Configuring Environment From the MATLAB Command Line” on page 4-21
- “Exporting and Importing Environment Properties” on page 4-31

### xPC Target Options Configuration Parameter

The configuration parameters **xPC Target Options** node appears when you select and apply one of the xPC Target options to the Simulink model **Configuration Parameter > Code Generation > System target file** parameter:

- `xpctarget.tlc`  
Generate code for an xPC Target target.
- `xpctargetert.tlc`  
Generate code for an xPC Target target using the required Embedded Coder™ software.

The **xPC Target Options** node allows you to specify how the software generates the target application. You might need to enter and select these options before you create (build) a target application. The default values of these options are reasonable for target application creation.

---

**Tip** If you set up your model to xPC Target Embedded Coder (`xpctargetert.tlc`), you can create a custom Code Replacement Library (CRL), which must be based upon the xPC Target BLAS (XPC\_BLAS). For more on CRLs, see:

- Code Replacement Library (CRL) and Embedded Targets
  - “Code Replacement”.
- 

See “Configuration Parameters” in the *xPC Target User’s Guide* for more information on the **xPC Target Options** node.



## xPC Target Explorer

In this section...
“Basic Operations” on page 4-3
“Menus, Toolbars, and Shortcut Keys” on page 4-6
“Default Target Computers” on page 4-8

### Basic Operations

xPC Target Explorer is a graphical user interface for the xPC Target product. It provides a single point of contact for almost all interactions. Through xPC Target Explorer, you can perform basic operations, such as

- Configure the host computer for the xPC Target software
- Add and configure target computers for the xPC Target software, up to 64 target computers
- Create boot CDs and removable drives for particular target computers
- Connect the target computers for your xPC Target system to the host computers
- Download a prebuilt target application, DLM, to a target computer
- Start and stop the application that has been downloaded to the target
- Add host, target, or file scopes to the downloaded target application
- Monitor signals
- Add signals to xPC Target scopes and remove them
- Start and stop scopes
- Adjust parameter values for the signals while the target application is running

The xPC Target Explorer GUI runs on your xPC Target system host computer.

---

**Note** xPC Target Explorer runs only on 32-bit host computers. On 64-bit computers, use the MATLAB command-line interface. For further information, see “Configuring Environment From the MATLAB Command Line” on page 4-21.

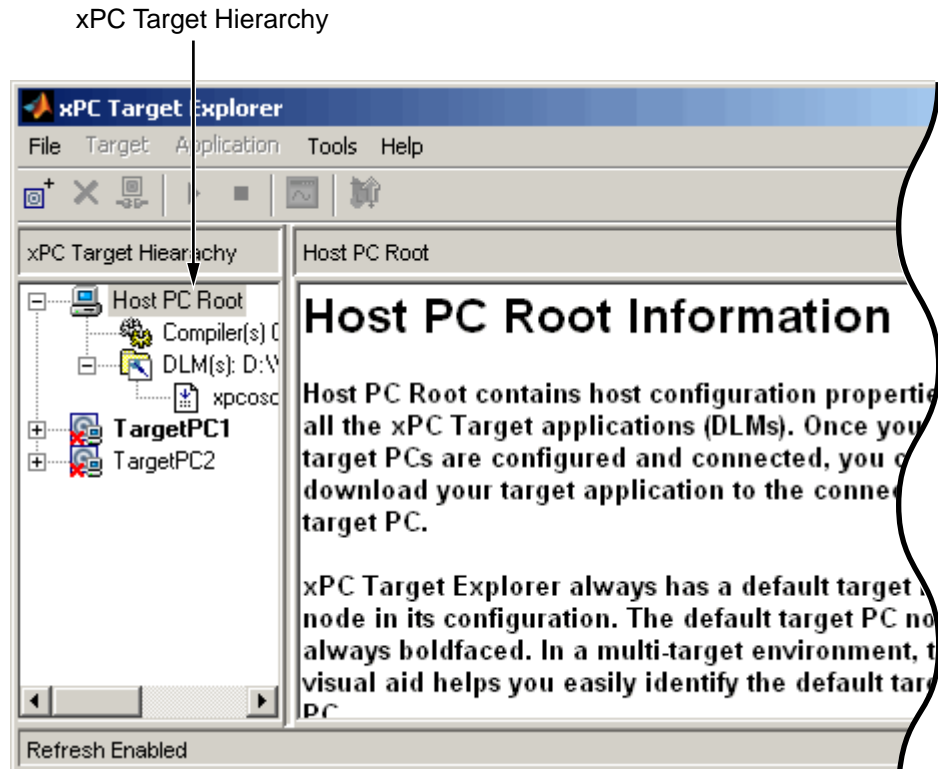
---

You can interact with xPC Target Explorer through menus or a toolbar. You can also right-click objects and select actions from the context menu for those objects. The tutorials in the xPC Target documentation describe procedures using mouse operations.

- 1 In the MATLAB Command Window, type

```
xpcexplr
```

The xPC Target Explorer window opens.



You can also start xPC Target Explorer from the Simulink model window (**Tools > Code Generation > xPC Target Explorer**).

You can dock or undock the xPC Target Explorer window using the arrow in the upper-right corner. Note the contents of the left pane of the xPC Target Explorer. This is the **xPC Target Hierarchy** pane. If you resize or move the window, the xPC Target software remembers the new size and location in subsequent restarts of xPC Target Explorer.

This pane contains all the objects in your xPC Target hierarchy. As you add objects to your system, xPC Target Explorer adds corresponding nodes to the **xPC Target Hierarchy** pane. The foremost node is the Host PC Root node.

It represents the host computer. The right pane displays information that reflects an item selected in the left pane.

Note that, by default, xPC Target Explorer starts with two target computer objects. The first target computer object is highlighted as the default.

---

**Note** Do not use Simulink external mode while xPC Target Explorer is running. Use only one interface or the other.

---

### Menus, Toolbars, and Shortcut Keys








The procedures in the xPC Target documentation for xPC Target Explorer use right-click operations. You can also perform xPC Target Explorer operations through the menu bar and toolbar. This section is a reference for the menu bar and toolbar contents.

The xPC Target Explorer menu bar has the following menus:

- **File** — General file operation options, including
  - **Add Target** — Add a target computer to the xPC Target system.
  - **Remove Target** — Remove a target computer from the xPC Target system.
  - **Change Host PC Current Directory** — Change folder to one that contains the prebuilt target application (DLM) you want to download to your target computers.
  - **Close** — Close xPC Target Explorer.
- **Target** — General target computer operations, including
  - **Ping Target** — Test the communication between the host computer and target computers.
  - **Connect to Target** — Connect xPC Target Explorer to selected target computer.
  - **Set As Default** — Designate the selected target computer as the default one.

- **Import Environment** — Import an existing target computer configuration from the MATLAB workspace.
- **Export Environment** — Save the target computer environment structure to a MAT-file.
- **Save Session** — Save target computer application sessions to xPC Target Explorer.
- **Load Session** — Load target computer application sessions to xPC Target Explorer.
- **Application** — General target application operations, including
  - **Start Application** — Start the selected target application.
  - **Stop Application** — Stop the selected target application.
  - **Add a scope** — Add a Host, Target, or File scope.
  - **Delete scope** — Delete a scope.
  - **View target computer scopes** — Display a viewer on the host computer for host scopes.
- **Tools** — General operations, including
  - **Enable/Disable Refresh** — Enable/disable the window refresh.
  - **Change Refresh Rate** — Change the refresh rate for the display of signals in xPC Target Explorer. In the Refresh Rate dialog box, enter the desired refresh rate in seconds. Note that setting the refresh rate to less than 0.2 seconds (default) might cause target computer CPU overloads or degrade MATLAB performance.
  - **Go to Simulink Model** — For the selected model, display the Simulink model.
- **Help** — General product help information, including
  - **Using xPC Target** — Invoke help for xPC Target product.
  - **xPC Target Explorer Help** — Invoke help for xPC Target Explorer.
  - **About xPC Target** — Invoke general xPC Target information.

The xPC Target Explorer toolbar has buttons for some of the more commonly used operations available from the menu bar. These buttons include

Button	Description
	Add <b>Target</b> button
	Delete <b>Target</b> button
	Connect to <b>Target</b> button
	Start <b>Application</b> button
	Stop <b>Application</b> button
	Scope <b>Viewer</b> button
	Go to <b>Simulink Model</b> button

The xPC Target Explorer has the following keyboard shortcuts:

Action	Shortcut
Add target	<b>Ctrl+A</b>
Remove target	<b>Ctrl+R</b>
Close	<b>Ctrl+W</b>
Stop/start target	<b>Ctrl+T</b>
Ping target	<b>Ctrl+P</b>
Delete scope	Select scope and click <b>Delete</b>

### Default Target Computers

The following are notes on default target computers:

- When you first start xPC Target Explorer, it has a default node, TargetPC1. You configure this node for a target computer, then connect the node to the target computer. If you later build a target application from a Simulink model, the xPC Target software builds and downloads that application for the default target computer. You can add other target computer nodes and designate one of them as the default target computer instead of the first one. To set a target computer node as the default, right-click that node and

select **Set As Default** from the context-sensitive menu. The default target computer node is always boldfaced. In a multitarget environment, this visual aid helps you easily see the target computer you are working with.

If you delete a default target computer node, the target computer node preceding it becomes the default target computer node. The last target computer node is always the default target computer node and cannot be deleted.

- If you want to use the xPC Target command-line interface to work with the target computer, you must indicate which target computer the command is interacting with. If you do not identify a particular target computer, the xPC Target software expects xPC Target Explorer to contain this information.
- The xPC Target product provides a default target computer to help you work with the MATLAB command-line interface, maintain compatibility with previous releases, and work with Simulink external mode, as follows:
  - When you define a default target computer, the MATLAB command-line interface works as in prior releases. For example, when you instantiate the target object constructor `xpctarget.xpc` without any arguments (for example, `tg=xpc`) the constructor uses the environment properties of the default target computer to communicate with the target computer.
  - The target computer environment object, `xpctarget.targets`, manages collective and individual target computer environments. See “Setting Up the Target Application Environment” on page 4-14 in the xPC Target user’s guide documentation for details.
  - The target computer commands `getxpcenv` and `setxpcenv` get and set environment properties for a single target computer system.

## Target Environment Command-Line Interface

In this section...
“Creating Target PC Environment Object Containers” on page 4-10
“Displaying Target PC Environment Object Property Values” on page 4-10
“Adding Target PC Environment Collection Objects” on page 4-11
“Removing Target PC Environment Collection Objects” on page 4-11
“Getting Target PC Environment Object Names” on page 4-11
“Changing Target PC Environment Object Defaults” on page 4-12
“Working with Particular Target PC Object Environments” on page 4-12

### Creating Target PC Environment Object Containers

`xpctarget.targets` is a container that manages target PC environment collection objects. To create an object container of type `xpctarget.targets`, use the constructor command `xpctarget.targets`. For example, the following creates a `tgs` object. In the MATLAB window, type

```
tgs = xpctarget.targets
```

The resulting target PC object container is `tgs` (target PC environment collection object) through which you can manage target PC environment objects.

### Displaying Target PC Environment Object Property Values

To display the properties of a target PC environment collection object, use the target PC object container method `xpctarget.targets.get` (`env` collection object). You can use either a method syntax or an object property syntax.

The syntax `get(env_collection_object)` can be replaced by

```
env_collection_object.get
```

In the MATLAB window, type



```
tgs.get
```

To display the value of particular target PC environment collection object property, use the syntax `get(env_collection_object, property_name)` or `env_collection_object.property_name`.

In the MATLAB window, type

```
tgs.DefaultTarget
```

## **Adding Target PC Environment Collection Objects**

To add a target PC environment collection object, use the target PC environment collection object method `xpctarget.targets.Add (env_collection_object)`. In the MATLAB window, type

```
tgs.Add
```

Check that an additional target PC environment collection object has been added. Type

```
tgs.get
```

## **Removing Target PC Environment Collection Objects**

To delete a target PC environment collection object, use the environment collection object method, `xpctarget.targets.Remove (env_collection_object)`, of the `tgs` object. In the MATLAB window, type

```
tgs.Remove('TargetPCName')
```

## **Getting Target PC Environment Object Names**

By default, each time you add a target PC environment object, xPC Target names that object with the string `TargetPCN`, where `N` increments with each subsequent target PC environment object with that base name.

To get a target PC environment object, use the target PC environment collection object method `xpctarget.targets.getTargetNames (env_collection_object)`. Type

```
tgs.getTargetNames
```

You can change a target PC environment object name through the xPC Target Explorer, or programmatically by setting the Name property of the environment object.

### **Changing Target PC Environment Object Defaults**

By default, the first target PC environment object is the default one. Functions such as `getxpcenv` and `setxpcenv` operate only on the default target PC environment object.

To make another environment object be the default one, use the target PC environment collection object method `xpctarget.targets.makeDefault` (env collection object). Type

```
tgs.makeDefault('TargetPC2')
```

### **Working with Particular Target PC Object Environments**

To manage the properties of a particular target PC object environment, use the target PC object collection environment method `xpctarget.targets.Item` (env collection object). This method retrieves an xPC Target environment object from the `xpctarget.targets` class. You can then assign this object to a variable and manipulate that object. Type

```
env2=tgs.Item('TargetPC2')
```

`env2` is now the target environment object for TargetPC2.

If you want to work with the default target PC object environment, use the `DefaultTarget` property. For example,

```
env=tgs.DefaultTarget
```

With the object variables, you can manage the target PC environment object properties. For example, to get the object properties, type

```
env2.get
```

Use the dot notation to change the property values. For example, to change the IP address of TargetPC2 to 192.168.0.10, the subnet mask to 255.255.255.0, type

```
env2.TcpIpTargetAddress='192.168.0.10'  
env2.TcpIpSubNetMask='255.255.255.0'
```

To check your changes, type

```
env2.get
```

Alternatively, you can type

```
env.TcpIpTargetPortenv2.TcpIpTargetAddress
```

## Setting Up the Target Application Environment

### In this section...

“Introduction” on page 4-14

“Getting a List of Environment Properties” on page 4-14

“Getting a List of Environment Properties for Single target computer Systems” on page 4-15

“Changing Environment Properties with xPC Target Explorer” on page 4-16

“Changing Environment Properties with a Command-Line Interface” on page 4-19

“Changing Environment Properties with a Command-Line Interface for Single Target Computer Systems” on page 4-20

### Introduction

The xPC Target environment defines the connections and communication between the host and target computers. It also defines the build process for a real-time application. You can define the xPC Target environment through either the MATLAB interface or xPC Target Explorer. The xPC Target environment provides a number of demos that help you understand the product.

See “Target Environment Command-Line Interface” on page 4-10 for a description of how you can manage multiple target computer environments through the MATLAB interface. Alternatively, if you have only one target computer, you can use the function `getxpcenv` to list the environment variables for the default target computer environment.

To enter properties specific to your model and its build procedure, see “Entering Simulation Parameters” in the *xPC Target Getting Started Guide*. 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 these properties. To get a list of the

property names, their allowed values, and their current values, for a specific target computer, TargetPC2.

In the MATLAB Command Window, type

```
tgs=xpctarget.targets  
tgEnv=tgs.Item('TargetPC2')
```

The MATLAB interface displays a list of xPC Target environment properties and the current values. For a list of the properties, see the function `xpctarget.env.get` (env object).

Alternatively, you can use the xPC Target Explorer window to view and change environment properties.

## Getting a List of Environment Properties for Single target computer Systems

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

**1** In the MATLAB Command Window, type

```
setxpcenv
```

The MATLAB interface displays a list of xPC Target environment properties and the allowed values. For a list of the properties, see the function `getxpcenv`.

**2** Type

```
getxpcenv
```

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

Alternatively, you can use the xPC Target Explorer window to view and change environment properties.

## Changing Environment Properties with xPC Target Explorer

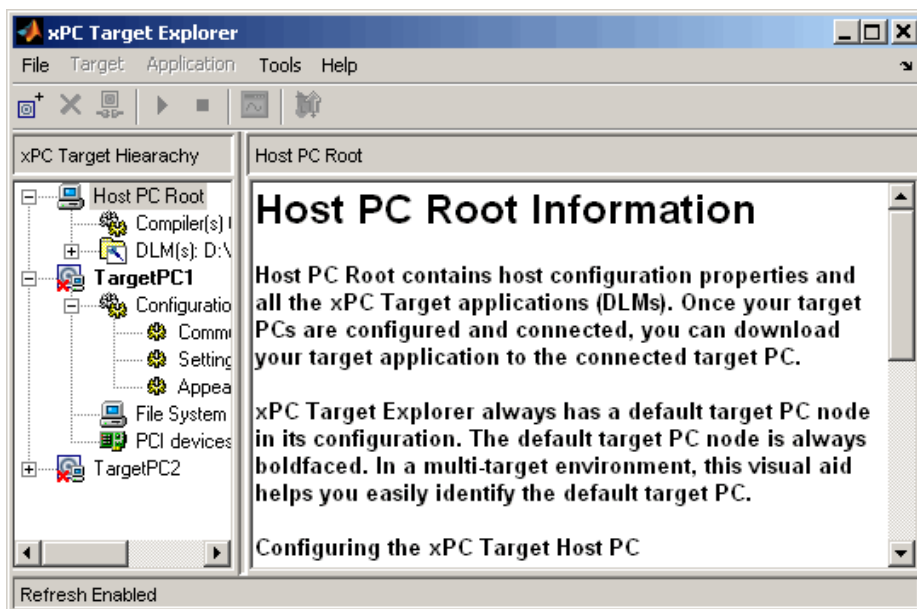
The xPC Target software lets you define and change environment properties. These properties include the path to the C/C++ compiler, the host computer COM port, the logging buffer size, and many others. Collectively these properties are known as the xPC Target environment.

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

- 1 In the MATLAB window, type

```
xpcexplr
```

The MATLAB interface opens the xPC Target Explorer window.



Note the contents of the left pane. This is the **xPC Target Hierarchy** pane.

This pane contains all the objects in your xPC Target hierarchy. As you add objects to your system, xPC Target Explorer adds their corresponding nodes to the **xPC Target Hierarchy** pane. The most important node is the **HostPC** node. It represents the host computer. The most important node is the **TargetPC** node. Each time you add a target computer node to xPC Target Explorer, a corresponding node is added to the **xPC Target Hierarchy** pane, starting with **TargetPC1** and incrementing with the addition of each new target computer node.

The right pane displays information about the item selected in the left pane. This pane also displays xPC Target environment properties for the **HostPC** and **TargetPC** nodes. You edit these properties in the right pane.

To change the size of the left or right pane, select and move the divider between the panes left or right.

The **Configuration** node under the **target PC** node has the target computer-specific configuration pane. If your license does not include the xPC Target Embedded Option™ product, you can choose **Boot Floppy**, **CD Boot**, **DOS Loader**, or **Network Boot**. With the xPC Target Embedded Option license, you have the additional choice of **Standalone**.

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

xPC Target Explorer applies changes to the environment properties as soon as you make them in the right pane.

To change environment properties for target computers, see “Configuring Environment Parameters for Target Computers” on page 4-17.

## Configuring Environment Parameters for Target Computers

You can optionally configure the environment parameters for the target computer node in your xPC Target system. This section assumes that

- You have already added target computer nodes to your system.
- You have already configured the communication parameters between the host computer and the target computer.

---

**Note** In general, you can run the xPC Target software using the default values of these parameters.

---

**1** In the xPC Target Explorer, expand a target computer node.

A Configuration node appears. Under this are nodes for Communication, Settings, and Appearance. The parameters for the target computer node are grouped in these categories.

**2** Select Settings.

The **Settings Component** pane appears to the right.

**3** In the **Target RAM size (MB)** field, enter

- Auto — The target kernel automatically attempts to determine the amount of memory.
- Manual — The amount of RAM, in MB, installed on the target computer.

This field defines the total amount of installed RAM in the target computer. The RAM is used for the kernel, target application, data logging, and other functions that use the heap.

---

**Note** The xPC Target kernel can use only 2 GB of memory.

---

**4** From the **Maximum model size** list, select either 1 MB, 4 MB, or 16 MB. Choosing the maximum model size reserves the specified amount of memory on the target computer for the target application. The remaining memory is used by the kernel and by the heap for data logging. Note that this parameter is only available for Standalone mode. You cannot specify a maximum model size for Boot Floppy, DOSLoader, or Network Boot modes. These modes allow the loading of arbitrarily-sized target applications.

---

**Note** You cannot build a 16 MB target application to run in Standalone mode.

---



- 5** By default, the **Enable secondary IDE** check box is not selected. Select this check box only if you want to use the disks connected to a secondary IDE controller. If you do not have disks connected to the secondary IDE controller, do not select this check box.
- 6** By default, the **Target PC is a 386/486** check box is not selected. You must select this check box if your target computer has a 386 or 486 compatible processor. If your target computer has a Pentium or higher compatible processor, selecting this check box will slow the performance of your target computer.
- 7** By default, the **Multicore CPU support** check box is cleared. You can select this check box if your target computer has multicore processors and you want to take advantage of them. If you also want to enable multirate models to take advantage of target computers, see “Configuring Models for Targets with Multicore Processors”.
- 8** In the **xPC Target Hierarchy** pane, select **Appearance**.  
  
The **Appearance Component** pane appears to the right.
- 9** From the **Target scope** list, select either **Enabled** or **Disabled**. The property **Target scope** is set by default to **Enabled**. If you set **Target scope** to **Disabled**, the target computer displays information as text. To use all the features of the target scope, you also need to install a keyboard on the target computer.
- 10** Set the **Target scope** property to **Enabled**.

## Changing Environment Properties with a Command-Line Interface

The xPC Target software 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 a MATLAB code script that accesses the environment settings according to your own needs. For example, you could write a script that switches between two targets.

To change the COM port property for your host computer from COM1 to COM2:

In the MATLAB window, type

```
tgs=xpctarget.targets  
tgEnv=tgs.Item('TargetPC2')  
set(tgEnv,'RS232HostPort','COM2')
```

### **Changing Environment Properties with a Command-Line Interface for Single Target Computer Systems**

The xPC Target software 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 a MATLAB code script that accesses the environment settings according to your own needs. For example, you could write a script that switches between two targets.

To change the COM port property for your host computer from COM1 to COM2:

In the MATLAB window, type

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

## Configuring Environment From the MATLAB Command Line

### In this section...

“Setup Requirements” on page 4-21

“Configuring the Compiler” on page 4-22

“Configuring Communications” on page 4-23

“Configuring Boot Methods” on page 4-25

### Setup Requirements

For regression test purposes or to run the xPC Target software on a 64-bit system, you can use the MATLAB command line interface to give xPC Target information about the software and hardware configuration. Use the `xpctarget.env.set` (env object) and target environment object (see “Target Environment Command-Line Interface” on page 4-10 in the xPC Target™ User’s Guide on page 1)

This topic provides the minimum configuration for running your target computer with the xPC Target environment. For a complete list of configuration properties, see `xpctarget.env.set` (env object).

---

**Note** To display the allowed values of xPC Target environment properties, type `setxpcenv` with no arguments. To display their current values, type `getxpcenv` with no arguments.

---

The command examples assume that you have one target computer to configure, TargetPC1. If you want to add another to your environment, type the following in the MATLAB Command Window:

```
tgs=xpctarget.targets;  
get(tgs);  
tgs.Add;  
get(tgs);
```

xPC Target adds and names the target by an increment of 1. To configure the new target, reference its name in your configuration commands, for example, TargetPC2.

For further information on working in the xPC Target environment from the MATLAB Command Window, see the following sections in the xPC Target™ User's Guide on page 1:

- Chapter 7, “Targets and Scopes in the MATLAB Interface”
- “Target Environment Command-Line Interface” on page 4-10

### Configuring the Compiler

On the host computer:

- 1 In the MATLAB Command Window, type:

```
xpcsetCC('setup')
```

This function queries the host computer for C compilers that the xPC Target environment supports. It returns output like the following:

```
Select your compiler for xPC Target.
```

```
[1] Microsoft Visual C++ Compilers 2008 Professional Edition (SP1) in  
    c:\Program Files (x86)\Microsoft Visual Studio 9.0  
[2] Microsoft Visual C++ Compilers 2010 Professional in  
    C:\Program Files (x86)\Microsoft Visual Studio 10.0  
  
[0] None
```

```
Compiler:
```

- 2 At the Compiler prompt, enter the number for the compiler that you want to use. For example, 2.

The function verifies your selection:

```
Verify your selection:
```

```
Compiler: Microsoft Visual C++ Compilers 2010 Professional
```

Location: C:\Program Files (x86)\Microsoft Visual Studio 10.0

Are these correct [y]/n?

**3** Type y or press **Enter** to verify the selection.

The function finishes the dialog.

Done...

Now you can proceed to “Configuring Communications” on page 4-23 for the xPC Target environment.

## Configuring Communications

Set the properties for the xPC Target environment according to the communications method that your host and target computers use:

- “Network Communications” on page 4-23
- “Serial Communications” on page 4-24

### Network Communications

On the host computer, set the properties that your host and target computers require for network connections.

**1** In the MATLAB Command Window, type the following commands to create and environment object for your target:

```
tgs=xpctarget.targets
tgs.makeDefault('TargetPC1')
env=tgs.Item('TargetPC1')
```

**2** Verify the following property settings:

Property	Value
HostTargetComm	TcpIp.
TcpIpTargetAddress	Target PC IP address, for example, 192.168.0.10.

<b>Property</b>	<b>Value</b>
TcpIpTargetPort	22222 (default).
TcpIpSubNetMask	Subnet mask address of your LAN, for example 255.255.255.0.
TcpIpGateway	Gateway IP address, 255.255.255.255 (default).
TcpIpTargetDriver	Ethernet driver. Auto (default) allows the software to choose the driver for the target computer system. If the driver is Auto and the bus type is USB, the software defaults the target driver to USBAX772, the driver most commonly used.
TcpIpTargetBusType	Target computer bus type, PCI (default) or USB.
TcpIpTargetISAMemPort and TcpIpTargetISAIRQ	If you are using an ISA bus Ethernet card, enter values for these properties. The values of these properties must correspond to the jumper settings or ROM settings on your ISA bus Ethernet card.

For more information about these settings, see “Network Communication”.

- 3** Update values as required. For example:

```
env.HostTargetComm='TcpIp'
```

When the communication properties are set, see “Configuring Boot Methods” on page 4-25.

## **Serial Communications**

---

**Note** RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.

---

On the host computer, set the properties that your host and target computers require for serial connections:

- 1 In the MATLAB Command Window, type the following commands to create and environment object for your target:

```
tgs=xpctarget.targets
tgs.makeDefault('TargetPC1')
env=tgs.Item('TargetPC1')
```

- 2 Verify the following property settings:

Property	Value
HostTargetComm	RS232.
RS232HostPort	COM1 (default).
RS232Baudrate	115200 (default).

For more information on these settings, see “Serial Communication”.

- 3 Update values as required. For example:

```
env.HostTargetComm='RS232'
```

When the communication properties are set, see “Configuring Boot Methods” on page 4-25.

## Configuring Boot Methods

Choose a boot method that your host and target computers support:

- “Booting Target Computers from CD or DVD with a Command-Line Interface” on page 4-26
- “Booting Target Computers Within a Dedicated Network with a Command-Line Interface” on page 4-28
- “Booting Target Computers from Removable Drives” on page 4-29

### Booting Target Computers from CD or DVD with a Command-Line Interface

You use a boot CD/DVD to load and run the xPC Target kernel. After you make changes to the xPC Target environment properties, you must create a CD/DVD bootable ROM. Before you start:

- Acquire an empty, writable CD or DVD.
- Acquire a CD/DVD-RW drive.
- Choose a process for creating a bootable CD or DVD. You can create a boot CD or DVD in one of the following ways:
  - The xPC Target software can create a boot CD or DVD for you. To use this capability, your host computer must have one of the following Windows® systems:
    - Microsoft® Windows 7
    - Microsoft Windows Vista™
    - Microsoft Windows XP Service Pack 2 or 3 with Image Mastering API v2.0 (IMAPIv2.0), available at <http://support.microsoft.com/kb/KB932716>.
  - Third-party CD/DVD writing software can write ISO image files for you. Use this method if you do not have Microsoft Windows 7, Microsoft Windows Vista, or Microsoft Windows XP Service Pack 2 or 3.

---

**Note** Standard Microsoft Windows software (such as Windows Explorer or Windows Media Player) does not write ISO image files to CD/DVD.

---

To create a boot CD/DVD for the TargetPC1 environment:

- 1 Insert the empty CD or DVD in the host computer.
- 2 In the MATLAB Command Window, type

```
tgs=xpctarget.targets
tgs.makeDefault('TargetPC1')
env=tgs.Item('TargetPC1')
```



**3** Verify the following property settings:

Property	Value
TargetBoot	CDBoot
CDBootImageLocation	Location of CD/DVD disk drive on host computer

**4** Update values as required. For example:

```
env.TargetBoot='CDBoot'
env.CDBootImageLocation='c:\work\xpc\cdimage'
```

**5** In the MATLAB Command Window, type:

```
xpcbootdisk
```

The xPC Target software displays the following message and creates the CD/DVD boot ISO image.

```
Current boot mode: CDBoot
CD boot image is successfully created
```

**6** Perform one of the following, depending on your software:

- If you have Microsoft Windows 7, Microsoft Windows Vista, or Microsoft Windows XP Service Pack 2 or 3 with Image Mastering API v2.0 (IMAPIv2.0), xpcbootdisk prompts you to insert a CD/DVD.

```
Insert an empty CD/DVD. Available drives:
[1] z:\
[0] Cancel Burn
```

Insert the CD or DVD in the drive, then press the **Enter** key.

- If you do not have Microsoft Windows 7, Microsoft Windows Vista, or Microsoft Windows XP Service Pack 2 or 3 with Image Mastering API v2.0 (IMAPIv2.0), use your third-party software to write the `cdboot.iso` image to the empty CD/DVD.

**7** When the CD/DVD drive stops, remove the CD/DVD.

Your next task is to install the software on the target computer and test your installation. See “Running the Confidence Test”.

### **Booting Target Computers Within a Dedicated Network with a Command-Line Interface**

If you boot target computers on a dedicated network, you do not need a boot drive. You do need to set up the host computer and target computer. Before you start using the command-line interface to perform this operation, see “Booting Target Computers Within a Dedicated Network”. On 64-bit MATLAB systems, instead of using xPC Target Explorer, use the command-line interface:

- 1** In the MATLAB Command Window, type:

```
tgs=xpctarget.targets  
tgs.makeDefault('TargetPC1')  
env=tgs.Item('TargetPC1')
```

- 2** Verify the following property setting:

<b>Property</b>	<b>Value</b>
TargetBoot	NetworkBoot

- 3** Update values as required. For example:

```
env.TargetBoot='NetworkBoot'
```

- 4** Set a TCP/IP address. Verify that the subnet of this IP address is the same as the host computer. Otherwise your network boot will fail. For example, type:

```
env.TcpIpTargetAddress='10.10.10.11'
```

- 5** Set the target computer MAC address (in hexadecimal).

```
env.TargetMACAddress='01:23:45:67:89:ab'
```


- 6** In the MATLAB Command Window, type:

```
xpcnetboot
```

The following message appears:

Current boot mode: NetworkBoot

The software creates and starts a network boot server process on the host computer. You will boot the target computer using this process.

A minimized icon () representing the network boot server process appears on the bottom right of the host computer system tray.

## Booting Target Computers from Removable Drives

You can use a removable boot drive to load and run the xPC Target kernel. The following kinds of drives are supported:

- USB flash drives
- SD (Compact) flash drives
- Removable hard drives
- 3.5-inch floppy disks.

After you make changes to the xPC Target environment properties, you must create or update the boot drive.

- 1 In the MATLAB Command Window, type:

```
tgs=xpctarget.targets
tgs.makeDefault('TargetPC1')
env=tgs.Item('TargetPC1')
```

Verify the following property settings:

Property	Value
TargetBoot	BootFloppy
BootFloppyLocation	Location of removable drive on host computer

- 2 Update values as required. For example:

```
env.TargetBoot='BootFloppy'
env.BootFloppyLocation='a:'
```

- 3** In the MATLAB Command Window, type:

```
xpcbootdisk
```

The following message appears:

```
Current boot mode: BootFloppy  
Insert a formatted floppy disk into your host PC's  
disk drive and press a key to continue
```

- 4** Insert a formatted removable drive in the host computer and then press any key.

The write procedure starts. While creating the boot drive, the MATLAB Command Window displays the following status information:

```
Creating xPC Target boot disk ... Please wait  
xPC Target boot disk successfully created.
```

Your next task is to install the software on the target computer and test your installation. See “Running the Confidence Test”.

## Exporting and Importing Environment Properties

The xPC Target Explorer consists of the property settings you define for the Configuration node, for example, for the host communication method and so forth. When you have settings that you are happy with, you can save them as variables in the MATLAB workspace.

This topic describes how to export target computer node property settings in a structured format to the MATLAB workspace. It assumes that you have set settings with xPC Target Explorer. See “Environment Properties for Serial Communication” or “Environment Properties for Network Communication”.

- 1 If the xPC Target Explorer is not open, open it now. At the MATLAB Command Window, type

```
xpcexplr
```

---

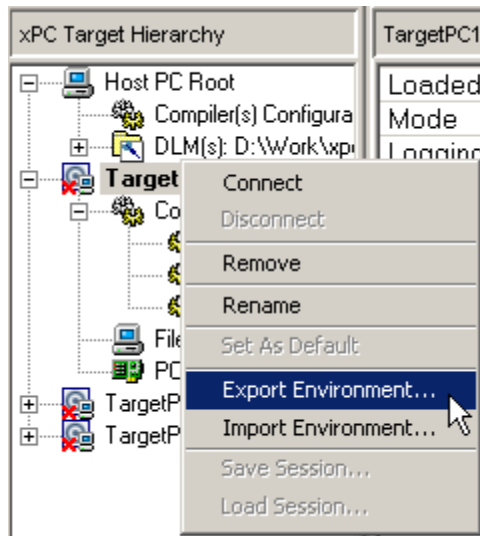
**Note** xPC Target Explorer runs only on 32-bit host computers. On 64-bit computers, use the MATLAB command-line interface. For further information, see “Configuring Environment From the MATLAB Command Line” on page 4-21.

---

- 2 In the xPC Target Explorer **xPC Target Hierarchy** pane, right-click the Configuration node of the target computer for which you want to save the configuration. For example, right-click TargetPC1.

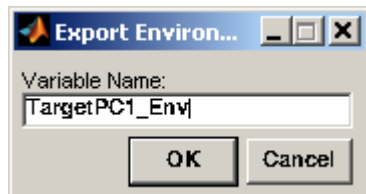
A context-sensitive menu is displayed.

- 3 Select **Export Environment**.



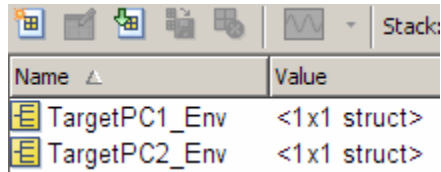
The Export Environment to Workspace dialog is displayed.

- 4 In the Export Environment to Workspace dialog box, enter a unique name. For example, type TargetPC1env.



- 5 Add configuration variables for as many target computer configurations as you need.

The following illustrates a MATLAB workspace with two xPC Target Explorer environment configuration variables.



Name	Value
TargetPC1_Env	<1x1 struct>
TargetPC2_Env	<1x1 struct>

You can save the target computer environment structure to a MAT-file. This enables you to reimport the configuration defined in the structure into a future xPC Target Explorer session.

- 1 To save the variable `TargetPC1env` in the MAT-file `targetpc1.mat`, in the MATLAB Command Window, type

```
save targetpc1.mat TargetPC1env
```

MATLAB saves the file `targetpc1.mat` in the current folder.

- 2 In the same MATLAB session, or in a different one, load the contents of `targetpc1.mat` into the MATLAB workspace. Type

```
load targetpc1.mat
```

- 3 If the xPC Target Explorer is not open, open it now.
- 4 In the xPC Target Explorer **xPC Target Hierarchy** pane, right-click the target computer node for which you want to import the configuration. For example, right-click `TargetPC1`.

A context-sensitive menu is displayed.

- 5 Select **Import Environment**.
- 6 In the Import Environment Structure dialog, select a previously exported configuration. For example, select `TargetPC1env`.





# Signals and Parameters

---

Changing parameters in your target application while it is running in real time, and checking the results by viewing signal data, are two important prototyping tasks. The xPC Target software includes command-line and graphical user interfaces to complete these tasks. This chapter includes the following sections:

- “Signal Monitoring” on page 5-2
- “Signal Tracing” on page 5-16
- “Signal Logging” on page 5-57
- “Parameter Tuning and Inlining Parameters” on page 5-66
- “Nonobservable Signals and Parameters” on page 5-86

## Signal Monitoring

In this section...
“Introduction” on page 5-2
“Signal Monitoring with xPC Target Explorer” on page 5-2
“Signal Monitoring with the MATLAB Interface” on page 5-9
“Monitoring Stateflow States” on page 5-10
“Animating Stateflow Charts” on page 5-14

### Introduction

Signal monitoring is the process for acquiring signal data during a real-time run without time information. The advantage with signal monitoring is that there is no additional load on the real-time tasks. Use signal monitoring to acquire signal data without creating scopes that run on the target computer.

In addition to signal monitoring, the xPC Target software enables you to monitor Stateflow states as test points through the xPC Target Explorer and MATLAB command-line interfaces. You designate data or a state in a Stateflow diagram as a test point. This makes it observable during execution. See the *Stateflow User's Guide* for details. You can work with Stateflow states as you do with xPC Target signals, such as monitoring or plotting Stateflow states.

After you start running a target application, you can use signal monitoring to get signal data.

---

**Note** xPC Target Explorer works with multidimensional signals in column-major format.

---

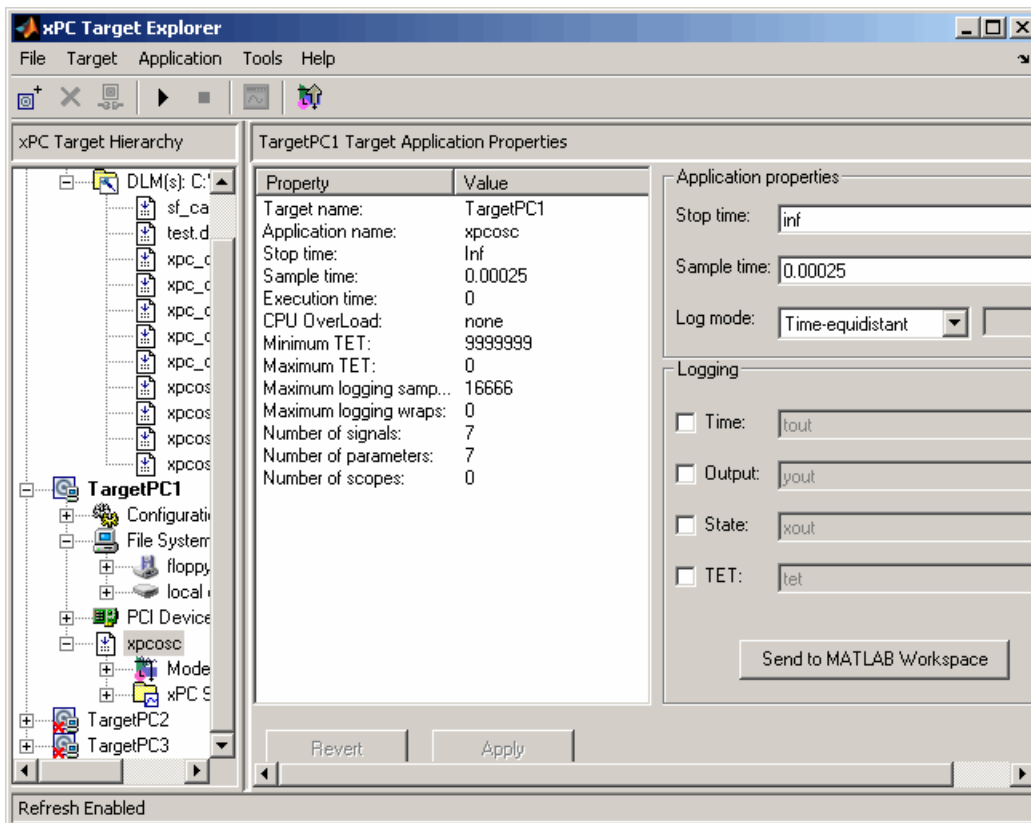
### Signal Monitoring with xPC Target Explorer

This procedure uses the model `xpcosc.mdl` as an example, and assumes you created and downloaded the target application to the target computer. For meaningful values, the target application should be running.

- 1 If xPC Target Explorer is not started, start it now. In xPC Target Explorer, select the node of the running target application in which you are interested, for example, xpcosc.

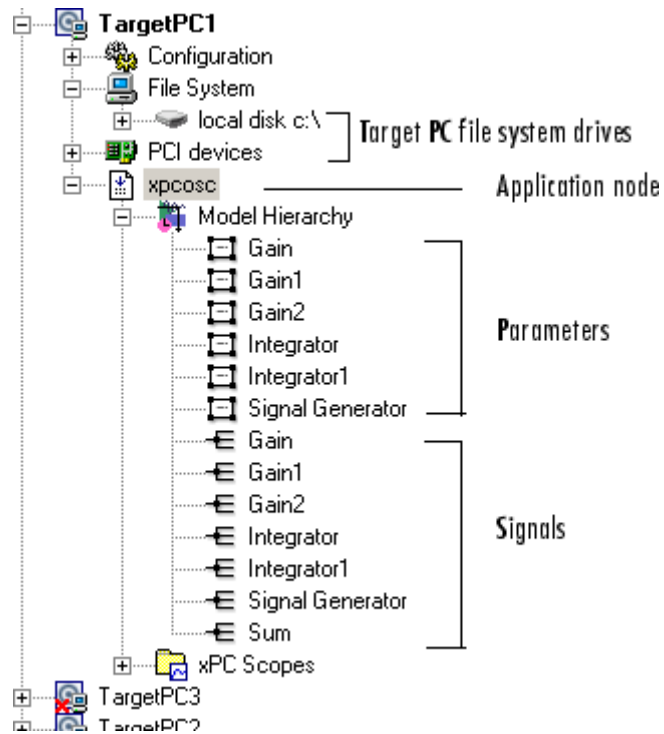
The target computer **Target Application Properties** pane appears.

- 2 In the **Solver** pane, change the **Stop time** parameter to `inf` (infinity). Click **Apply**.








- 3 To get the list of signals in the target application, expand the target application node, then expand the Model Hierarchy node under the target application node.

The model hierarchy expands to show the Simulink objects (signals and parameters) in the Simulink model.



The Model Hierarchy node can have the following types of nodes:

Icons	Nodes
	Subsystems, including their signals and parameter
	Referenced models, including their signals set as test points
	Parameters
	Signals
	Stateflow states set as test points

Only xPC Target tunable parameters and signals of the application, as represented in the Simulink model, appear in the Model Hierarchy node.

---

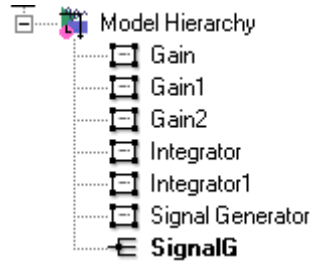
**Note** This example currently has only parameters and signals. If a block has a name that consists of only spaces, xPC Target Explorer does not display a node for that block. To monitor a signal from that block, provide an alphanumeric name for that block and rebuild and download that block.

---

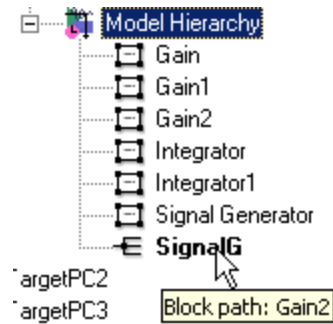
If you make changes (such as adding an xPC Target scope) to the model associated with the downloaded application, then rebuild that model and download it again to the target computer, you should reconnect to the target computer to refresh the Model Hierarchy node.

- 4 To view only labeled signals (the xPC Target software refers to Simulink signal names as signal labels) :
  - a Open the `xpcosc.mdl` file.
  - b Right-click a signal line and name that signal. For example, right-click the output of the Signal Generator block and name it `SignalG`.
  - c Build and download the updated model.
  - d When the updated model is displayed in xPC Target Explorer, right-click the Model Hierarchy node and select **View Only Labeled Signals**. This command assumes that you have labeled one or more signals in your model.

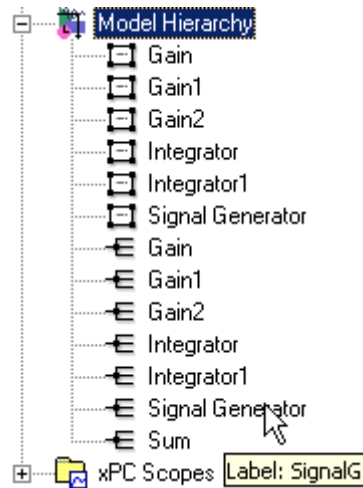
- e Re-expand the Model Hierarchy node to see the labeled signals.



To view the block path for a labeled signal, hover over the labeled signal. For example,



To display all the model signals again, right-click the Model Hierarchy node and select **View All Signals**. You can still view the signal label by hovering over the labeled signal. For example,



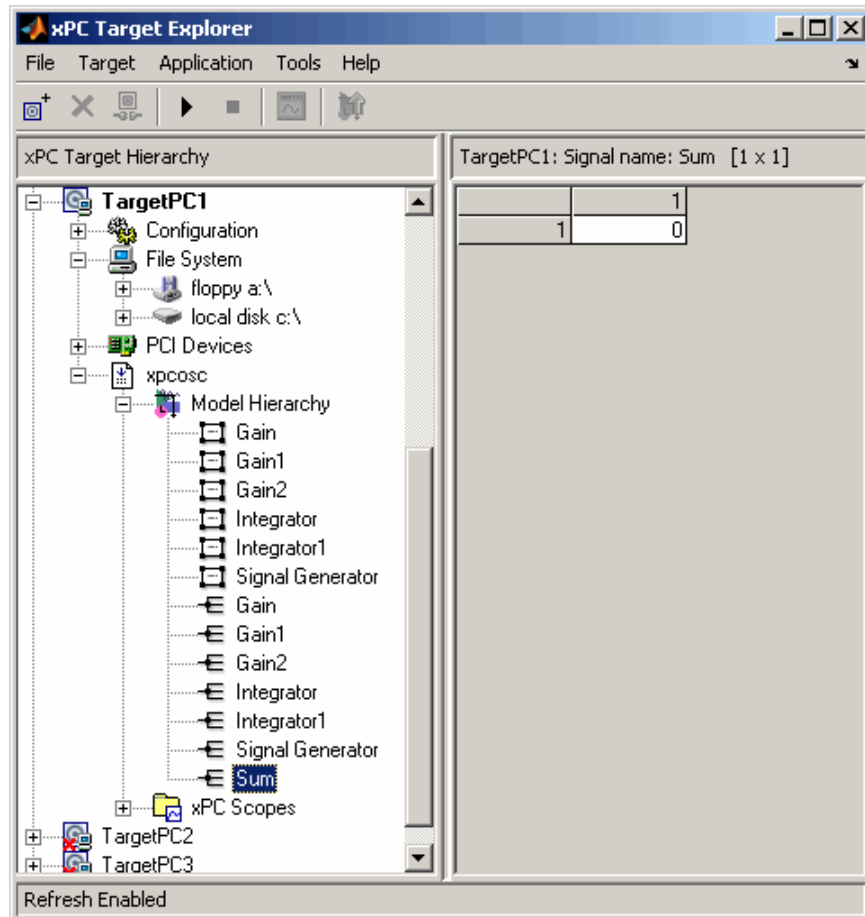

---

**Note** When working with signal labels:

- Signal labels must be unique.
  - xPC Target software ignores signal labels in referenced models.
- 

- f** Return to the model, remove the signal name you added, and rebuild and download the target application. The remaining examples in this section assume that you do not have any labelled signals in your model.
- 5** To go to the corresponding Simulink model subsystem, right-click the application node and select **Go to Simulink subsystem or block**.
- 6** To get the value of a signal, select the signal in the Model Hierarchy node.

The value of the signal is shown in the right pane.



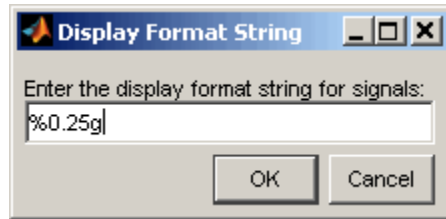
- 7 Right-click the target application and select **Start**.

The application starts running.

- 8 To change the numeric format display of the signal, right-click the Model Hierarchy node and select **Edit Signals Format String**.

The Display Format String dialog box is displayed.





- 9 Enter the signal format. Use one of the following. By default, the format is %0.25g.

Type	Description
%e or %E	Exponential format using e or E
%f	Floating point
%g	Signed value printed in f or e format depending on which is smaller
%G	Signed value printed in f or E format depending on which is smaller

### Monitoring Signals from Referenced Models

You can monitor signals from referenced models the same way that you do any other signal, with the exception that you must set the test point for the signal in the referenced model before you can monitor it. Additionally, the software ignores signal labels in referenced models.

### Signal Monitoring with the MATLAB Interface

This procedure uses the model `xpc_osc3.mdl` as an example, and assumes you created and downloaded the target application to the target computer. It also assumes that you have assigned `tg` to the target computer.

- 1 To get a list of signals, type either

```
set(tg, 'ShowSignals', 'on')
```

or

```
tg.ShowSignals='on'
```

The latter command causes the MATLAB window to display a list of the target object properties for the available signals. For example, the signals for the model `xpc_osc3.mdl` are shown below. Note that the `Label` column is empty because there are no labelled signals in the model. If your signal has a unique label, its label is displayed in this column. If the label is not unique, the command returns an error. If the signal label is in a referenced model, the software ignores it.

```
ShowSignals = on
  Signals = INDEX  VALUE                BLOCK NAME          LABEL
              0      0.000000          Signal Generator
              1      0.000000          Transfer Fcn
```

- 2 To get the value of a signal, use the `getsignal` method. In the MATLAB Command Window, type

```
tg.getsignal(0)
```

where 0 is the signal index. the MATLAB interface displays the value of signal 1.

```
ans=
    3.731
```

---

**Note** The xPC Target software lists referenced model signals with their full block path. For example, `xpc_osc5/childmodel/gain`.

---

See also “Signal Tracing with the MATLAB Interface” on page 5-40.

## Monitoring Stateflow States

This procedure uses the model `old_sf_car.mdl` as an example. It describes one way to set Stateflow states as test points for monitoring.

- 1 In the MATLAB window, type

```
old_sf_car
```

- 2 In the Simulink window, click **Simulation > Configuration Parameters**.

The Configuration Parameters dialog box is displayed for the model.

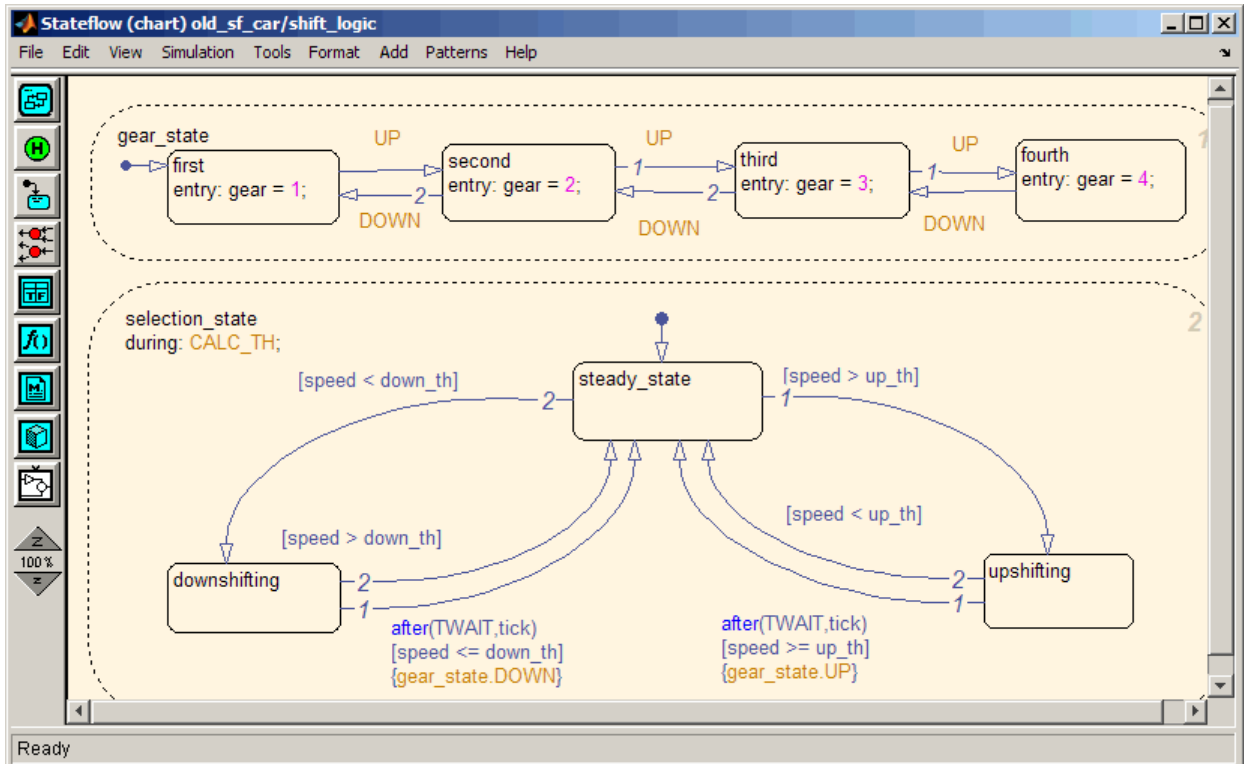
- 3** Click the **Code Generation** node.

The code generation pane opens.

- 4** To build a basic target application, in the **Target selection** section, click the **Browse** button at the **System target file** list. Click `xpctarget.tlc`, then click **OK**.
- 5** To make further changes, click the **xPC Target options** node.
- 6** When you are done, click **OK**.

7 In the old\_sf\_car model, double-click the shift\_logic chart.

The shift\_logic chart is displayed.



8 In the chart, click **Tools > Explore**.

The Model Explorer is displayed.

9 In the Model Explorer, expand old\_sf\_car.

10 Expand shift\_logic.

11 Expand gear\_state, then select first.

12 In the rightmost pane, **State first**, select the **Test point** check box. This creates a test point for the first state.

**13** Click **Apply**.

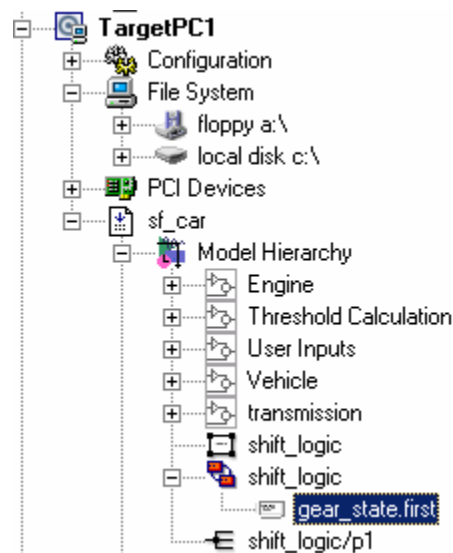
**14** Build and download the `old_sf_car` target application to the target computer.

You can now view the states with xPC Target Explorer or the MATLAB interface.

### Monitoring Stateflow States with xPC Target Explorer

This topic assumes that you have already set Stateflow states as test points (see “Monitoring Stateflow States” on page 5-10 if you have not).

- 1** If the xPC Target Explorer is not yet started, start it now and connect it to the target computer that has the downloaded `old_sf_car` target application.
- 2** To view the test point in the xPC Target Explorer, expand the Model Hierarchy node and navigate to `shift_logic`. The test point `gear_state.first` appears like any other signal in the hierarchy, as follows:



- 3 Choose the state as you do a signal to monitor.

### Monitoring Stateflow States with the MATLAB Interface

This topic assumes that you have already set Stateflow states as test points (see “Monitoring Stateflow States” on page 5-10 if you have not).

- 1 To get a list of signals in the MATLAB Command Window, type

```
tg=xpc
```

- 2 To display the signals in the target application, type either

```
set(tg, 'ShowSignals', 'on'); tg
```

or

```
tg.ShowSignals='on'
```

The latter causes the MATLAB window to display a list of the target object properties for the available signals.

For Stateflow states that you have set as test points, the state appears in the `BLOCK NAME` column like any signal. For example, if you set a test point for the `first` state of `gear_state` in the `shift_logic` chart of the `old_sf_car` model, the state of interest is `first`. This state appears as follows in the list of signals in the MATLAB interface:

```
shift_logic:gear_state.first
```

`shift_logic` is the path to the Stateflow chart and `gear_state.first` is the path to the specific state.

### Animating Stateflow Charts

The xPC Target software supports the animation of Stateflow charts in your model to provide visual verification that your chart behaves as expected.

This topic assumes that you are familiar with the use of Stateflow animation. For more information on Stateflow animation, see “Animating Stateflow Charts” in the Stateflow documentation.

---

The following describes steps that are particular to animating Stateflow charts for xPC Target systems.

- 1** Select **Simulation > External** in the Simulink window.
- 2** Select **Tools > External Mode Control Panel** in the Simulink window.  
The **External Signal & Triggering** window for the model is displayed.
- 3** In the Trigger section of the **External Signal & Triggering** window:
  - Set **Mode** to normal.
  - In the **Duration** field, enter 5.
  - Select the **Arm when connecting to target** check box.
- 4** In the Simulink window, from the **Simulation** menu, click **Configuration Parameters**.
- 5** Navigate to the xPC Target options node.
- 6** Select the **Enable Stateflow animation** check box.
- 7** Build and download the model to the target computer.
- 8** In the Simulink window, from the **Simulation** menu, click **Connect to Target**.
- 9** In the Simulink window, from the **Simulation** menu, click **Start Real-Time Code**.
- 10** To observe the animation, open the Stateflow Editor for your model.

---

**Note** Enabling the animation of Stateflow charts also displays additional Stateflow information. The Stateflow software requires this information to animate charts. You can disregard this information.

---

## Signal Tracing

In this section...
“Introduction” on page 5-16
“Signal Tracing with xPC Target Explorer” on page 5-16
“Signal Tracing with the MATLAB Interface” on page 5-40
“Signal Tracing with xPC Target Scope Blocks” on page 5-49
“Signal Tracing with Simulink External Mode” on page 5-51
“Signal Tracing with a Web Browser” on page 5-55

### Introduction

Signal tracing is the process of acquiring and visualizing signals while running a target application. In its most basic sense, allows you to acquire signal data and visualize it on the target computer or upload the signal data and visualize it on the host PC while the target application is running.

Signal tracing differs from signal logging. With signal logging you can only look at signals after a run is finished and the log of the entire run is available. For information on signal logging, see “Signal Logging” on page 5-57.

---

**Note** xPC Target Explorer works with multidimensional signals in column-major format.

---

### Signal Tracing with xPC Target Explorer

The procedures in this topic use the model `xpcosc.mdl` as an example, and assume you have created, downloaded, and started the target application on the target computer.

- “Creating Scopes” on page 5-17
- “Adding Signals to Scopes” on page 5-24
- “Stopping Scopes” on page 5-28



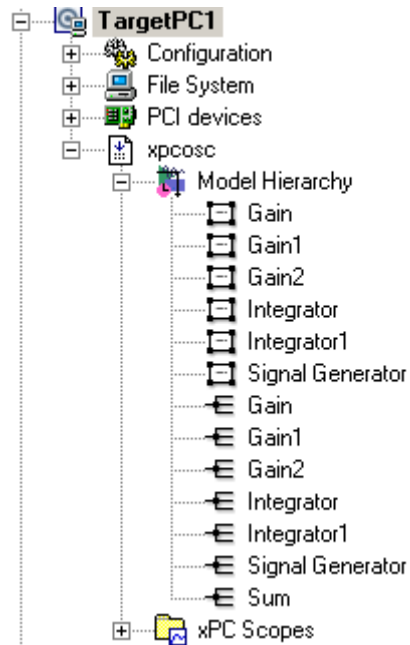
- “Software Triggering Scopes” on page 5-29
- “Configuring the Host Scope Viewer” on page 5-31
- “Acquiring Signal Data into Multiple, Dynamically Named Files on the Target Computer” on page 5-31
- “Copying Files to the Host Computer” on page 5-35
- “Exporting Data from File Scopes to MATLAB Workspace” on page 5-36
- “Saving and Reloading xPC Target Application Sessions” on page 5-38
- “Deleting Files from the Target Computer” on page 5-40

You can add or remove signals from target or host scopes while the scope is either stopped or running. For file scopes, you must stop the scope first before adding or removing signals.

## Creating Scopes

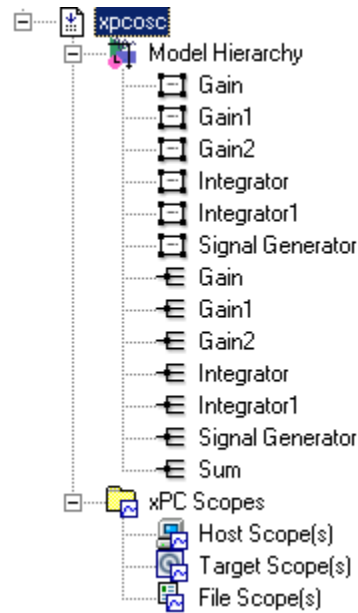
- 1** In xPC Target Explorer, verify that your xpcosc application is still running.
- 2** To get the list of signals in the target application, expand the `Model Hierarchy` node under the target application.

The model hierarchy expands to show the elements in the Simulink model.



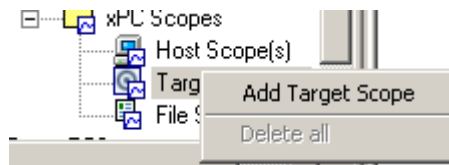
- 3 To get the list of scope types you can have in the target application, expand the xPC Scopes node below the application node.

The xPC Scopes node expands to show the possible scope types a target application can have.



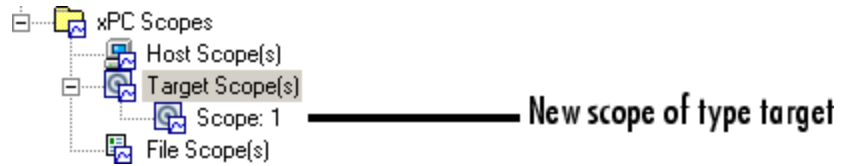
- 4** To create a scope to display on the target computer, right-click the Target Scopes node under the xPC Scopes node.

A context menu appears. This lists the actions you can perform on target computer scopes. For example, within the current context, you can create a target computer scope.



- 5** Select **Add Target Scope**.

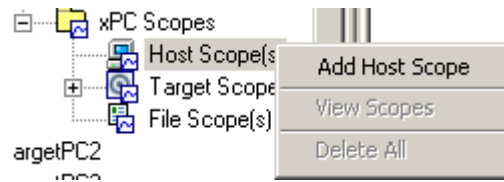
A scope node appears under Target Scopes. In this example, the new scope is labeled Scope 1.



You can add other scopes, including those of type `host` and `file`. Note, you can add as many file and host scopes as you want, as long as your target computer resources can support them.

- 6 To create a scope to be displayed on the host computer, right-click the `Host Scopes` node under the `xPC Scopes` node.

A context menu appears. This lists the actions you can perform on host computer scopes. For example, within the current context, you can create a host computer scope.

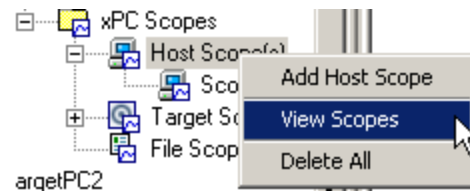


- 7 Select **Add Host Scope**.

A scope node appears under `Host Scopes`. In this example, the new scope is labeled as `Scope 2`.

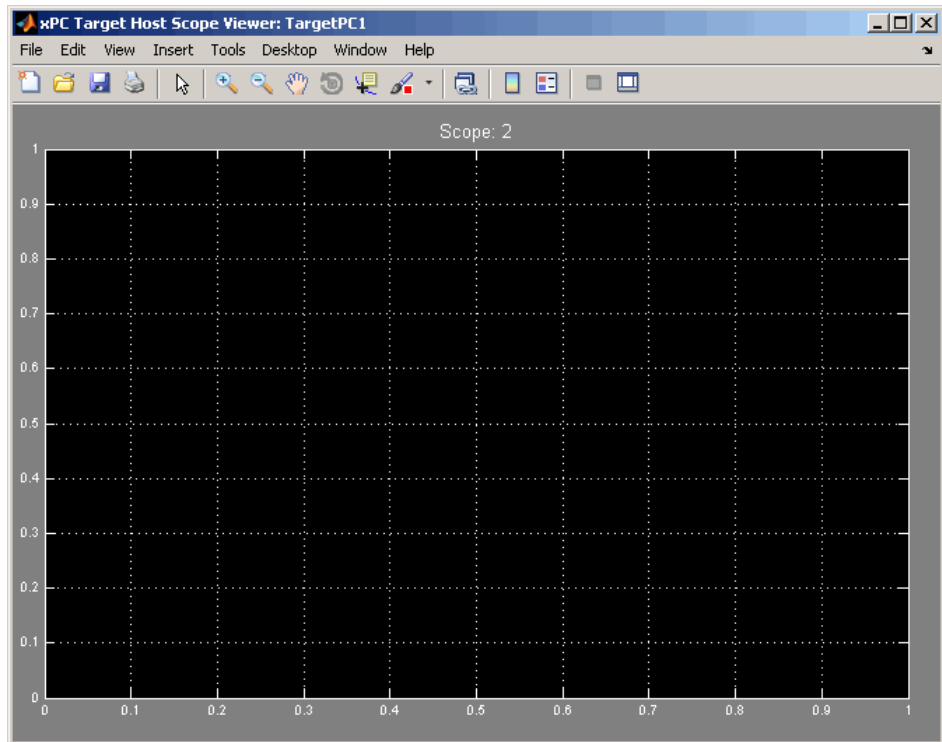
- 8 To visualize the host scope on the host computer, right-click `Host Scopes` from the `xPC Scopes` node.

A drop-down list appears.



## 9 Select View Scopes.

The xPC Target Host Scope Viewer window appears. Note that the signals you add to the scope will appear at the top right of the viewer.

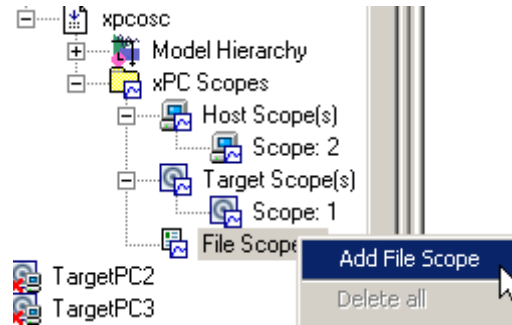


- 10 To list the properties of the scope object `Scope_2`, click the xPC Target Explorer tab to return to that window, and left-click `Scope_2`. (Note that you can configure the docking view using the MATLAB docking feature.)

The scope properties are shown in the rightmost pane.

- 11 To create a scope to acquire signal data into a file on the target computer file system, right-click the **File Scopes** node under the **xPC Scopes** node. Select **Add File Scope**.

A scope node appears under **File Scopes**. In this example, the new scope is labeled **Scope 3**.



By default, the software creates a file in the target computer **C:\** folder. The name of the file typically consists of the scope object name, **ScopeId**, and the beginning letters of the first signal added to the scope.

- 12** If you want to specify a different folder or filename, select the scope. The scope property pane is displayed. In the **File name** field, enter the full pathname for the file. Note that the current folder for the target computer appears at the top of the pane.

Property	Value
Target name:	TargetPC1
Application name:	xpcosc
ID:	3
Type:	File
Status:	Interrupted
Start time:	
Number of samples:	250
Decimation:	1
Number of Pre/Post Sa...	0
Trigger mode:	FreeRun
Trigger level:	0
Trigger slope:	Either
Trigger scope:	3
Trigger sample:	0
File name:	C:\sc1SigGen.dat
FAT Mode:	Lazy
Write Size:	512
Auto Restart:	off
Auto File Increment:	off
Max Write File Size:	536870912

**Scope data**

Number of samples:

Decimation:

Number of pre/post samples:

Trigger mode:

**Signal triggering**

Trigger level:

Trigger slope:

**Scope triggering**

Trigger scope:

Trigger sample:

**File settings**

File name:

(FAT) entry mode:

Write size:

Enable auto restart

Enable file auto increment

Max file size:

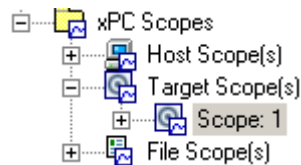
Your next task is to add signals to the scopes. The following procedure assumes that you have added scopes to the target computer and host computer.

### Adding Signals to Scopes

This topic describes how to add signals using the xPC Target Explorer **Add to Scopes** command. If a scope does not exist, you can drag a signal to a Host Scope, Target Scope, or File Scope icon to create a scope of that type in xPC Target Explorer.

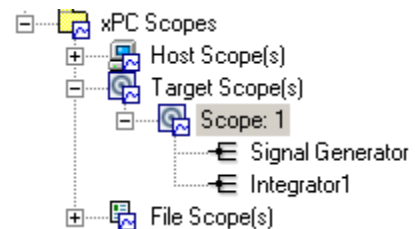
- 1 In the xPC Target Explorer window, add signals to the target computer scope, Scope 1. For example, to add signals `Integrator1` and `Signal Generator`, right-click each signal and select **Add to Scopes**. From the **Add to Scopes** list, select Scope 1. (Note that you can also drag the signal to the scope to add the signal to that scope.)

The Scope 1 node is shown with a plus sign.



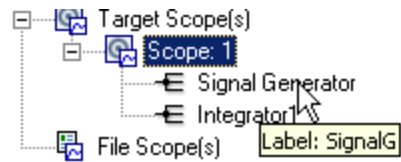
- 2 Expand the Scope 1 node.

The Scope 1 signals are displayed.

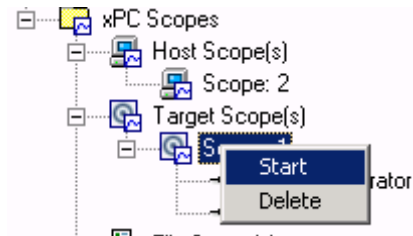


If one of the signals has been labeled, you can hover over the signal to display the signal label. For example,

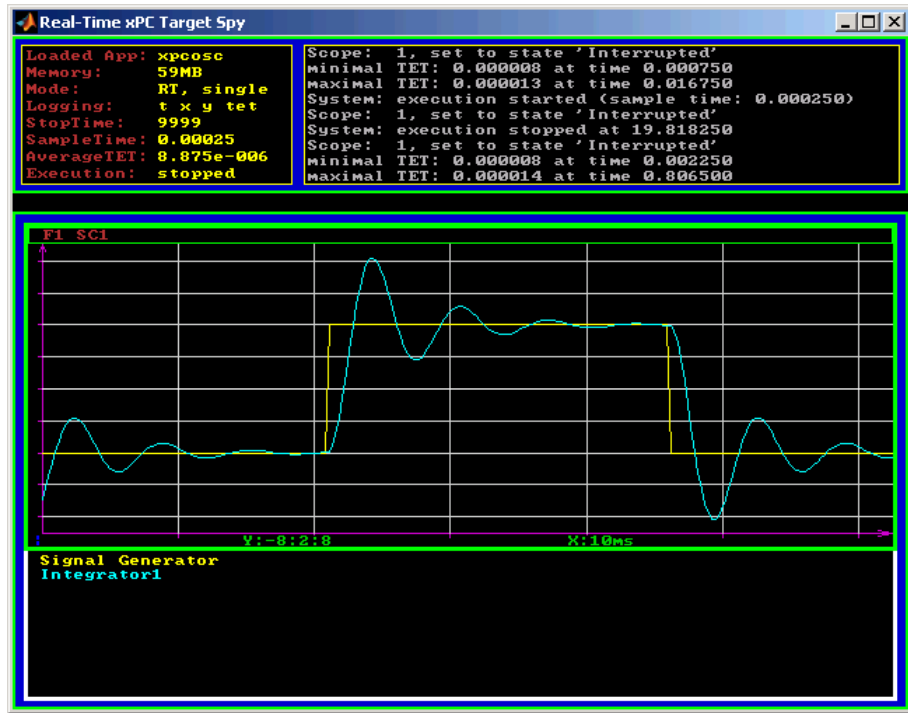




- 3 Start the scope. For example, to start Scope 1, right-click it and select **Start**.



The target screen plots the signals after collecting each data package. During this time, you can observe the behavior of the signals while the scope is running.

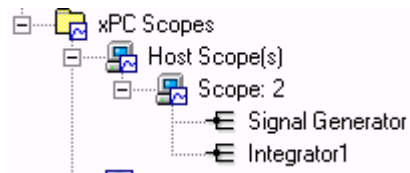


- 4 Add signals to the host computer scope. For example, to add signals Integrator1 and Signal Generator, right-click each signal and select **Add to Scopes**. From the **Add to Scopes** list, select Scope 2. (Note that you can also drag a signal from one scope to another to add that signal to another scope.)

The Scope 2 node is shown with a plus sign.

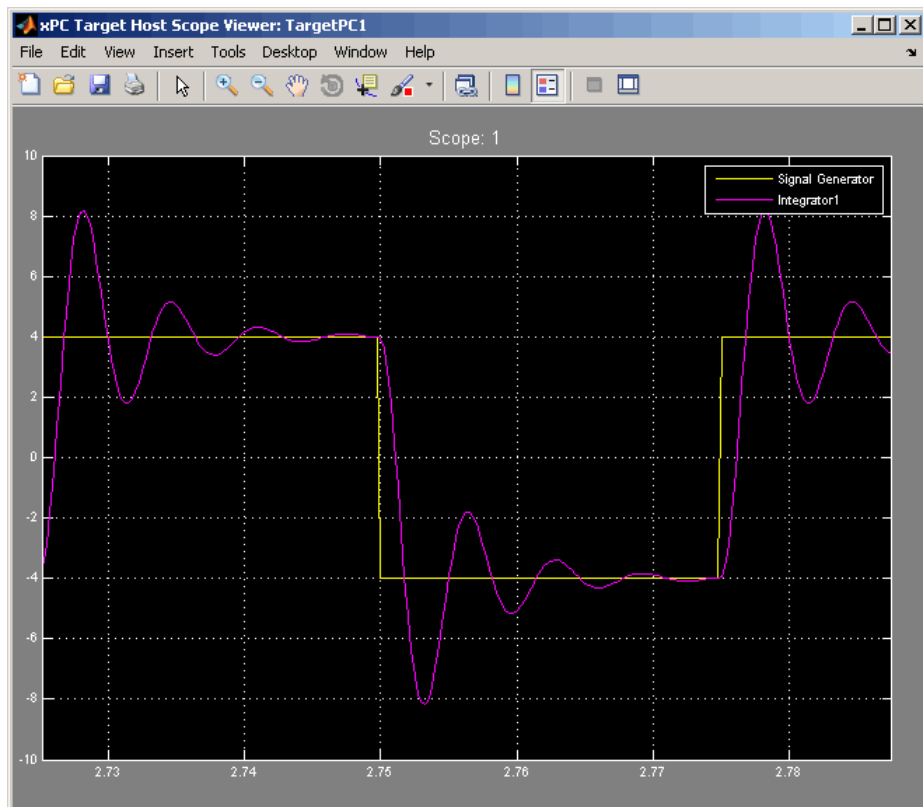
- 5 Expand the Scope 2 node.

The Scope 2 signals are displayed.



- 6 Start the scope. For example, to start the scope Scope 2, right-click **Scope 2** in the Host Scopes node of the xPC Target Explorer and select **Start**.

The xPC Target Host Scope Viewer window plots the signals after collecting each data package. During this time, you can observe the behavior of the signals while the scope is running.

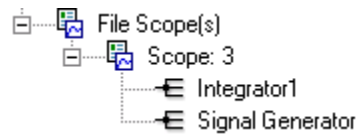


- 7 Add signals to the file scope. For example, to add signals `Integrator1` and `Signal Generator`, right-click each signal and select **Add to Scopes**. From the **Add to Scopes** list, select `Scope 3`. (Note that you can also drag a signal from one scope to another to add that signal to another scope.)

The `Scope 3` node is shown with a plus sign.

- 8 Expand the `Scope 3` node.

The `Scope 3` signals are displayed.



- 9 To specify a filename for the data file, select the file scope. In the right pane, enter a name in the **Filename** parameter. While in the parameter field, press **Enter** to save the filename.

Note that there is no name initially assigned. If you do not specify a filename, then after you start the scope, the software assigns a name for the target computer file to acquire the signal data. This name typically consists of the scope object name, `ScopeId`, and the beginning letters of the first signal added to the scope.

- 10 Start the scope. For example, to start the scope `Scope 3`, right-click **Scope 3** in the `File Scopes` node of the xPC Target Explorer and select **Start**.

For file scopes, the xPC Target software saves data to a file on the target computer flash disk.

Your next task is to stop the scopes. The following procedure assumes that you have started scopes.

### Stopping Scopes

- 1 Stop the scopes. For example, to stop `Scope 1`, right-click it and select **Stop**.

The signals shown on the target computer stop updating while the target application continues running, and the target computer displays the following message:

```
Scope: 1, set to state 'interrupted'
```

- 2 Stop the target application. For example, to stop the target application `xpcosc`, right-click `xpcosc` and select **Stop**.

The target application on the target computer stops running, and the target computer displays the following messages:

```
minimal TET: 0.0000006 at time 0.001250  
maximal TET: 0.0000013 at time 75.405500
```

Note that if you stop the target application before stopping the scope, the scope stops, too.

If you have file scopes, you can copy the file that contains the signal data from the target computer to the host computer. See “Copying Files to the Host Computer” on page 5-35.

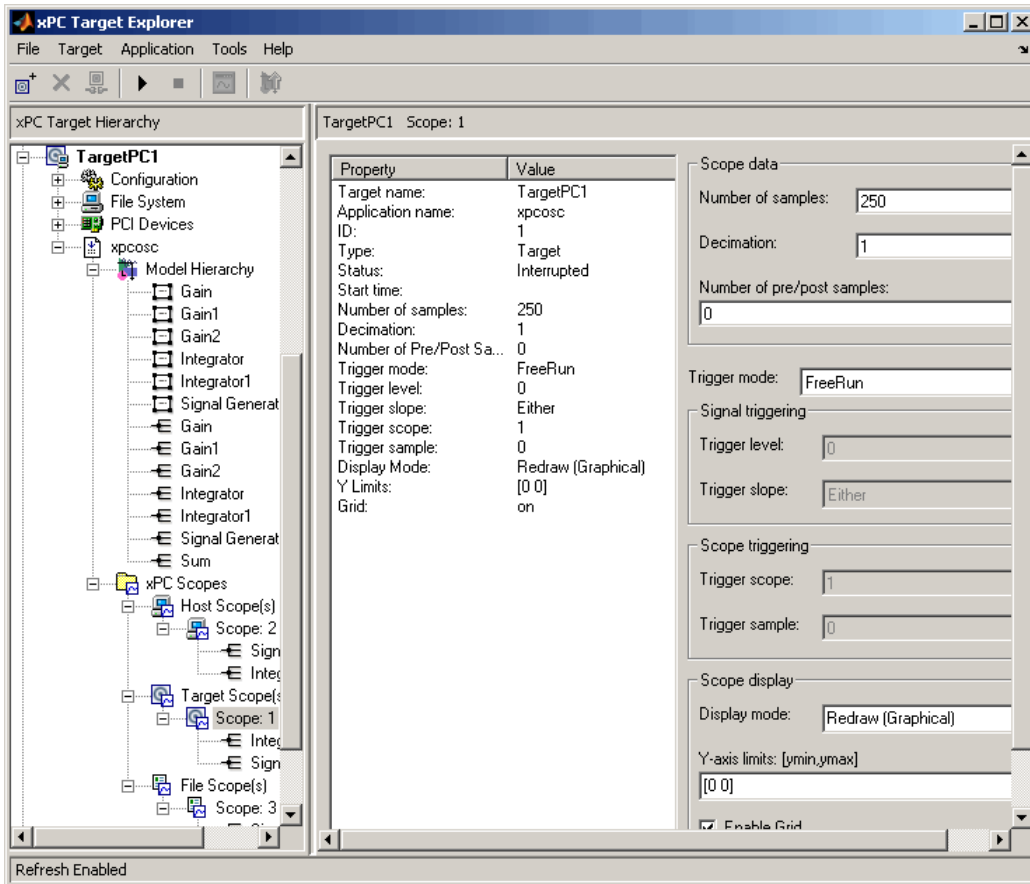
## Software Triggering Scopes

You can set up a scope such that only a user can trigger the scope. This section assumes that you have added a scope to your target application (see “Creating Scopes” on page 5-17) and that you have added signals to that scope (see “Adding Signals to Scopes” on page 5-24).

- 1 In the xPC Target Explorer window, select the scope that you want to trigger by software. For example, select Scope 1.

The properties pane for that scope is displayed.

2 From the **Trigger mode** list, select Software. Click **Apply**.



3 Start the scope and target application.

4 Observe that the scope has no plotted data.

5 Right-click the scope to be triggered. For example, select Scope 1.

6 Select Trigger.

7 Observe that the scope now has plotted data.

## Configuring the Host Scope Viewer

You can customize your host scope viewer. This section assumes that you have added a host scope to your target application, started the host scope viewer, and added signals `Integrator1` and `Signal Generator` (see “Creating Scopes” on page 5-17 and “Adding Signals to Scopes” on page 5-24). These viewer settings are per scope.

In the xPC Target Host Scope Viewer, right-click anywhere in the axis area of the viewer.

A context menu is displayed. This context menu contains options for the following:

- **View Mode** — Select **Graphical** for a graphical display of the data. Select **Numerical** for a numeric representation of the data.
- **Legends** — Select and deselect this option to toggle the display of the signals legend in the top right of the viewer.
- **Grid** — Select and deselect this option to toggle the display of grid lines in the viewer.
- **Y-Axis** — Enter the desired values. In the **Enter Y maximum limit** and **Enter Y minimum limit** text boxes, enter the range for the y-axis in the Scope window.
- **Export** — Select the data to export. Select **Export Variable Names** to export scope data names. In the **Data Variable Name** and **Time Variable Name** text boxes, enter the names of the MATLAB variables to save data from a trace. Click the **OK** button. The default name for the data variable is `application_name_scn_data`, and the default name for the time variable is `application_name_scn_time` where *n* is the scope number. Select **Export Scope Data** to export scope data to the MATLAB interface.

## Acquiring Signal Data into Multiple, Dynamically Named Files on the Target Computer

You can acquire signal data into multiple, dynamically named files on the target computer. For example, you can acquire data into multiple files to examine one file while the scope continues to acquire data into other files. To acquire data in multiple files, add a file scope to the application. After you build an application and download it to the target computer, you can add a

file scope to that application. You can then configure that scope to log signal data to multiple files. This section assumes that you have added a scope to your target application (see “Creating Scopes” on page 5-17). It also assumes that you have added signals to that scope (see “Adding Signals to Scopes” on page 5-24).

- 1** In xPC Target Explorer, expand the target computer node associated with the target computer file system you want to access. For example, expand TargetPC1.
- 2** Under TargetPC1, expand the target application node and navigate to the File Scope(s) node.
- 3** Right-click this node and add a file scope.



- 4** Add one or more scopes to that file scope. For example, add Integrator1 and Signal Generator to the scope.
- 5** Right-click the scope you just added (for example, Scope:1). The scope property pane for this scope is displayed.



Property	Value
Target name:	TargetPC1
Application name:	xpcosc
ID:	1
Type:	File
Status:	Interrupted
Start time:	
Number of samples:	250
Decimation:	1
Number of Pre/Post Sa...	0
Trigger mode:	FreeRun
Trigger level:	0
Trigger slope:	Either
Trigger scope:	1
Trigger sample:	0
File name:	unset
FAT Mode:	Lazy
Write Size:	512
Auto Restart:	off
Auto File Increment:	off
Max Write File Size:	536870912

Scope data

Number of samples:

Decimation:

Number of pre/post samples:

Trigger mode:

Signal triggering

Trigger level:

Trigger slope:

Scope triggering

Trigger scope:

Trigger sample:

File settings

File name:

(FAT) entry mode:

Write size:

Enable auto restart

Enable file auto increment

Max file size:

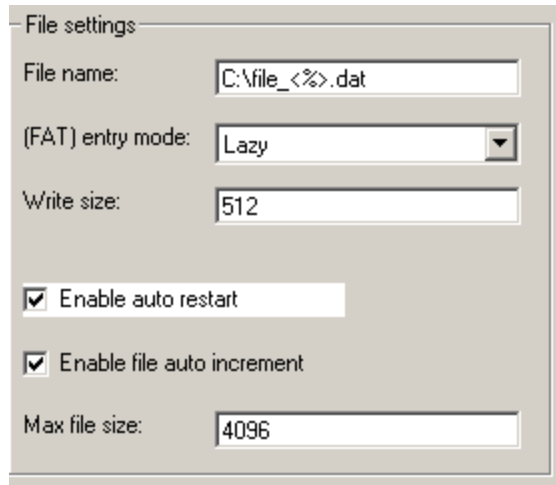
- 6 To enable the file scope to create multiple log files with the same name pattern, in the **File name** box, enter a name like `file_<%>.dat`.

This sequence directs the software to create up to nine log files, `file_1.dat` to `file_9.dat`, on the target computer file system.

- 7 Select the **Enable auto restart** check box. The **Enable file auto increment** check box is enabled.
- 8 Select the **Enable file auto increment** check box.
- 9 In the **Max file size** box, enter a value to limit the size of the signal log files. For example, to limit each log file size to 4096 bytes, enter 4096.

This value must be a multiple of the **Write size** value.

- 10 Click **Apply**. The saved values appear as follows:



- 11 Right-click the new file scope and select **Start**.
- 12 Start the associated target application.

The software creates a log file named `file_1.dat` and writes data to that file. When the size of `file_1.dat` reaches 4096 bytes (value of **Max file size**), the software closes the file. It then creates `file_2.dat` for writing until its size reaches 4096 bytes. The software repeats this sequence until it fills the last log file, `file_9.dat`. If the target application continues to run and collect data after `file_9.dat`, the software reopens `file_1.dat` and continues to log data, overwriting the existing contents. It cycles through the other log files sequentially.

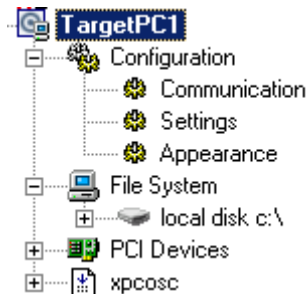
You can enable the creation of up to 99999999 files (<%%%%%%%%>.dat). The length of a file name, including the specifier, cannot exceed eight characters. See the **Filename** description in the `xpctarget.xpc.get` (target application object) for a details about this specifier.

## Copying Files to the Host Computer

From xPC Target Explorer, you can download target computer files from the target computer to the host computer.

- 1 In xPC Target Explorer, expand the target computer node associated with the target computer file system you want to access. For example, expand TargetPC1.
- 2 Under TargetPC1, expand the File System node.

Nodes representing the drives on the target computer are displayed.

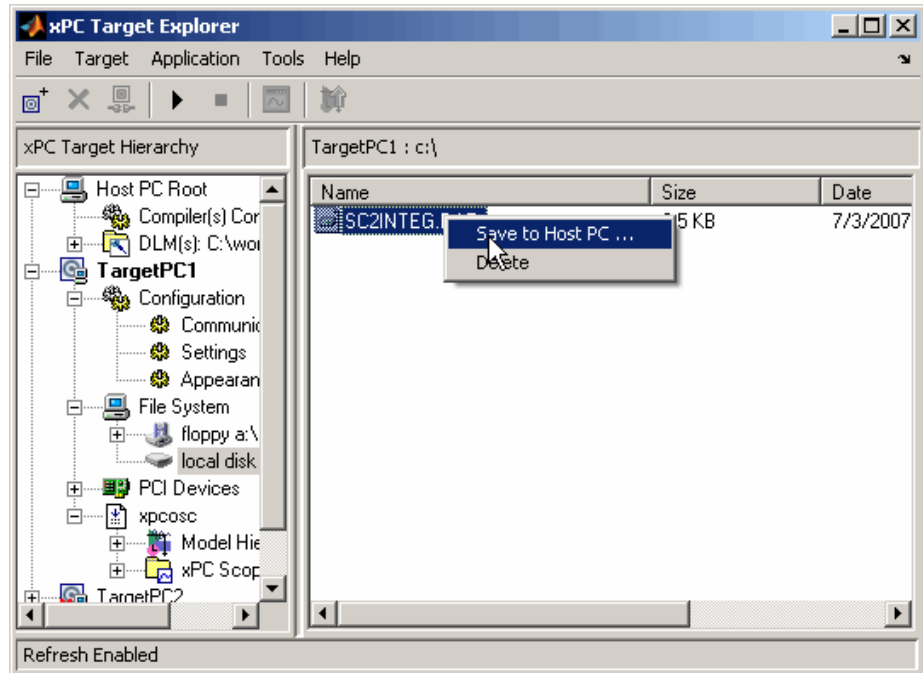


- 3 Expand the node of the drive that contains the file you want. For example, local disk: c:\.
- 4 Select the folder that contains the file you want. For example, select the node labeled local disk: c:\.

The contents of that folder are displayed in the right pane.

- 5 In the right pane, right-click the file you want to copy to the host computer. For example, right-click SC3SIGNA.DAT.

A context-sensitive menu is displayed.

6 Select **Save to host computer**.

A browser dialog box is displayed.

## 7 Choose the folder to contain the signal data file. If you want, you can also save the file under a different name or create a new folder for the file.

xPC Target Explorer copies the contents of the selected file, SC1INTEG.DAT for example, to the selected folder.

You can examine the contents of the signal data file. See “Retrieving a File from the Target Computer to the Host Computer” on page 8-8 in Chapter 8, “Logging Signal Data with FTP and File System Objects”.

### Exporting Data from File Scopes to MATLAB Workspace

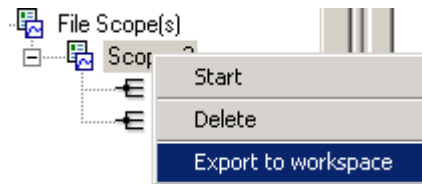
From xPC Target Explorer, you can export data from target computer files from the target computer to the MATLAB workspace. This topic assumes that

you have created a file scope that contains signal data (see “Adding Signals to Scopes” on page 5-24).

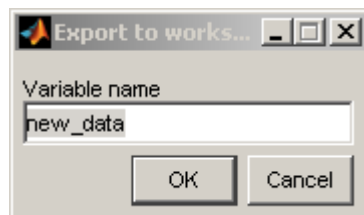
- 1 In xPC Target Explorer, expand the target computer node associated with the target computer file system you want to access. For example, expand TargetPC1.
- 2 Under TargetPC1, stop the target application.
- 3 Under TargetPC1, expand the xPC Scopes node.

All the scopes you have added are displayed.

- 4 Right-click on the file scope for which you want to export the signal data and select **Export to workspace**.

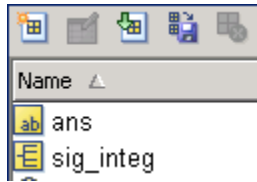


The **Export to workspace** dialog box is displayed.



- 5 Enter a variable name for the workspace data. For example, enter sig\_integ. Click **OK**.

In the MATLAB desktop, the **Workspace** pane displays the new variable name.



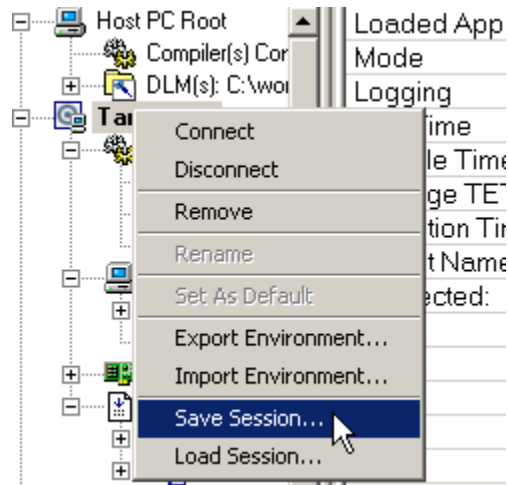
You can examine and otherwise manipulate the data. You can also plot the data with `plot(sig_integ.data)`. This is an alternate method to “Retrieving the Contents of a File from the Target Computer to the Host Computer” on page 8-12 in Chapter 8, “Logging Signal Data with FTP and File System Objects”.

### **Saving and Reloading xPC Target Application Sessions**

Once you have a set of application configurations, you can save the xPC Target application session, including scope and target computer settings, to a standard MATLAB MAT-file on the host computer. You can then later reload this saved session to another xPC Target application session. This feature lets you save and restore xPC Target application sessions so that you do not have to reconfigure target computer and target application settings each time you start xPC Target Explorer.

To save a session,

- 1** Connect xPC Target Explorer to a target computer.
- 2** In xPC Target Explorer, load a target application on the target computer.
- 3** In xPC Target Explorer, right-click the target computer you are interested in and select **Save Session**. For example,

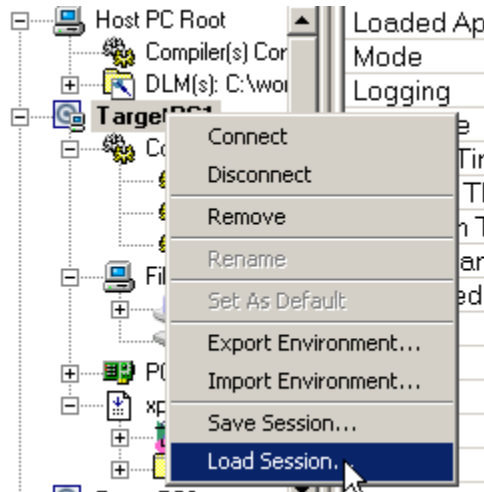


A **Save Target Session** as dialog box is displayed.

- 4 Browse to the desired folder and enter a unique name. For example, `xpcsession1.mat`.

To restore a session,

- 1 Connect xPC Target Explorer to a target computer.
- 2 In xPC Target Explorer, load a target application on the target computer. This target application must be the same target application for which the session was saved.
- 3 In xPC Target Explorer, right-click the target computer you are interested in and select **Load Session**. For example,



A **Load Target Session as** dialog box is displayed.

- 4 Browse to the desired folder and select the session you are interested in. For example, `xpcsession1.mat`.

A dialog box is displayed asking you to confirm that you want to load a new session.

- 5 Click **Yes**.

xPC Target Explorer loads the saved settings.

### Deleting Files from the Target Computer

From xPC Target Explorer on the host computer, you can delete the scope data file on the target computer file system. Use the same procedure as described in “Copying Files to the Host Computer” on page 5-35, but select **Delete** instead of **Save to Host PC**. xPC Target Explorer removes the selected file from the target computer file system.

### Signal Tracing with the MATLAB Interface

Creating a scope object allows you to select and view signals using the scope methods. This section describes how to trace signals using xPC Target



functions instead of using the xPC Target graphical user interface. This procedure assumes that you have assigned `tg` to the target computer.

After you create and download the target application, you can view output signals.

Using the MATLAB interface, you can trace signals with

- Host or target scopes (see “Signal Tracing with the MATLAB Interface and Target Scopes” on page 5-41 for a description of with target scopes)
- File scopes (see “Signal Tracing with the MATLAB Interface and File Scopes” on page 5-45)

You must stop the scope before adding or removing signals from the scope.

### **Signal Tracing with the MATLAB Interface and Target Scopes**

This procedure uses the Simulink model `xpcosc.mdl` as an example, and assumes you have built the target application for this model. It describes how to trace signals with target scopes.

**1** Start running your target application. Type any of

```
+tg
```

```
or
```

```
tg.start
```

```
or
```

```
start(tg)
```

The target computer displays the following message.

```
System: execution started (sample time: 0.0000250)
```

**2** To get a list of signals, type either

```
set(tg, 'ShowSignals', 'on')
```

```
or
```

```
tg.ShowSignals='on'
```

The MATLAB window displays a list of the target object properties for the available signals. For example, the signals for the model `xpcosc.mdl` are as follows:

```
ShowSignals = on
  Signals = INDEX  VALUE           BLOCK NAME      LABEL
            0      0.000000      Integrator1
            1      0.000000      Signal Generator
            2      0.000000      Gain
            3      0.000000      Integrator
            4      0.000000      Gain1
            5      0.000000      Gain2
            6      0.000000      Sum
```

For more information, see “Signal Monitoring with the MATLAB Interface” on page 5-9.

- 3 Create a scope to be displayed on the target computer. For example, to create a scope with an identifier of 1 and a scope object name of `sc1`, type

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

or

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

- 4 List the properties of the scope object. For example, to list the properties of the scope object `sc1`, type

```
sc1
```

The MATLAB window displays a list of the scope object properties. Notice that the scope properties `Time` and `Data` are not accessible with a target scope.

```
xPC Scope Object
  Application      = xpcosc
  ScopeId          = 1
  Status           = Interrupted
  Type             = Target
```

```
NumSamples           = 250
NumPrePostSamples    = 0
Decimation           = 1
TriggerMode          = FreeRun
TriggerSignal        = -1
TriggerLevel         = 0.000000
TriggerSlope         = Either
TriggerScope         = 1
TriggerSample        = -1
Mode                 = Redraw (Graphical)
YLimit               = Auto
Grid                 = On
Signals              = no Signals defined
```

- 5** Add signals to the scope object. For example, to add Integrator1 and Signal Generator, type

```
sc1.addsignal ([0,1])
```

or

```
addsignal(sc1,[0,1])
```

The target computer displays the following messages.

```
Scope: 1, signal 0 added
```

```
Scope: 1, signal 1 added
```

After you add signals to a scope object, the signals are not shown on the target screen until you start the scope.

- 6** Start the scope. For example, to start the scope sc1, type either

```
+sc1
```

or

```
sc1.start
```

or

```
start(sc1)
```

The target screen plots the signals after collecting each data package. During this time, you can observe the behavior of the signals while the scope is running.

**7** Stop the scope. Type either

```
-sc1
```

or

```
sc1.stop
```

or

```
stop(sc1)
```

The signals shown on the target computer stop updating while the target application continues running, and the target computer displays the following message.

```
Scope: 1, set to state 'interrupted'
```

**8** Stop the target application. In the MATLAB window, type either

```
-tg
```

or

```
tg.stop
```

or

```
stop(tg)
```

The target application on the target computer stops running, and the target computer displays the following messages.

```
minimal TET: 0.000023 at time 1313.789000  
maximal TET: 0.000034 at time 407.956000
```

## Signal Tracing with the MATLAB Interface and File Scopes

This procedure uses the Simulink model `xpcosc.mdl` as an example, and assumes you have built the target application for this model. It also assumes that you have a serial communication connection. This topic describes how to trace signals with file scopes .

---

**Note** The signal data file can quickly increase in size. You should examine the file size between runs to gauge the growth rate of the file. If the signal data file grows beyond the available space on the disk, the signal data might be corrupted.

---

- 1 Create an xPC Target application that works with file scopes. Type

```
tg=xpctarget.xpc('rs232', 'COM1', '115200')
```

- 2 To get a list of signals, type either

```
set(tg, 'ShowSignals', 'on')
```

or

```
tg.ShowSignals='on'
```

The MATLAB window displays a list of the target object properties for the available signals. For example, the signals for the model `xpcosc.mdl` are shown below.

```
ShowSignals = on
Signals =
```

INDEX	VALUE	BLOCK NAME	LABEL
0	0.000000	Integrator1	
1	0.000000	Signal Generator	
2	0.000000	Gain	
3	0.000000	Integrator	
4	0.000000	Gain1	
5	0.000000	Gain2	
6	0.000000	Sum	

For more information, see “Signal Monitoring with the MATLAB Interface” on page 5-9.

- 3** Start running your target application. Type

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

The target computer displays the following message:

```
System: execution started (sample time: 0.0000250)
```

- 4** Create a scope to be displayed on the target computer. For example, to create a scope with an identifier of 2 and a scope object name of `sc2`, type

```
sc2=tg.addscope('file', 2)  
or  
sc2=addscope(tg, 'file', 2)
```

- 5** List the properties of the scope object. For example, to list the properties of the scope object `sc2`, type

```
sc2
```

The MATLAB window displays a list of the scope object properties. Notice that the scope properties `Time` and `Data` are not accessible with a target scope.

```
xPC Scope Object  
Application      = xpcosc  
ScopeId         = 2  
Status          = Interrupted  
Type            = File  
NumSamples      = 250  
NumPrePostSamples = 0  
Decimation      = 1  
TriggerMode     = FreeRun
```

```
TriggerScope      = 2
TriggerSample     = 0
TriggerSignal     = -1
TriggerLevel      = 0.000000
TriggerSlope      = Either
ShowSignals       = off
FileName          = unset
Mode              = Lazy
WriteSize         = 512
AutoRestart       = off
DynamicFileName   = off
MaxWriteFileSize  = 536870912
```

Note that there is no name initially assigned to `FileName`. After you start the scope, xPC Target assigns a name for the file to acquire the signal data. This name typically consists of the scope object name, `ScopeId`, and the beginning letters of the first signal added to the scope.

- 6 Add signals to the scope object. For example, to add `Integrator1` and `Signal Generator`, type

```
sc2.addsignal ([4,5])
```

or

```
addsignal(sc2,[4,5])
```

The target computer displays the following messages.

```
Scope: 2, signal 4 added
```

```
Scope: 2, signal 5 added
```

After you add signals to a scope object, the file scope does not acquire signals until you start the scope.

- 7 Start the scope. For example, to start the scope `sc2`, type

```
+sc2
```

or

```
sc2.start
```

or

```
start(sc2)
```

The MATLAB window displays a list of the scope object properties. Notice that `FileName` is assigned a default filename to contain the signal data for the file scope. This name typically consists of the scope object name, `ScopeId`, and the beginning letters of the first signal added to the scope.

```
Application          = xpcosc
  ScopeId            = 2
  Status             = Pre-Acquiring
  Type               = File
  NumSamples         = 250
  NumPrePostSamples = 0
  Decimation         = 1
  TriggerMode       = FreeRun
  TriggerScope      = 2
  TriggerSample     = 0
  TriggerSignal     = 4
  TriggerLevel      = 0.000000
  TriggerSlope      = Either
  ShowSignals       = on
  Signals            = 4 : Integrator1
                    5 : Signal Generator
  FileName          = c:\sc7Integ.dat
  Mode              = Lazy
  WriteSize         = 512
  AutoRestart       = off
  DynamicFileName   = off
  MaxWriteFileSize  = 536870912
```

**8** Stop the scope. Type

```
-sc2
```

or

```
sc2.stop
```

or



```
stop(sc2)
```

- 9 Stop the target application. In the MATLAB window, type

```
-tg
```

or

```
tg.stop
```

or

```
stop(tg)
```

The target application on the target computer stops running, and the target computer displays messages similar to the following.

```
minimal TET: 0.00006 at time 0.004250  
maximal TET: 0.000037 at time 14.255250
```

To access the contents of the signal data file that the file scope creates, use the xPC Target file system object (`xpctarget.fs`) from the host computer MATLAB window. To view or examine the signal data, you can use the `readxpcfile` utility with the `plot` function. For further details on the `xpctarget.fs` file system object and the `readxpcfile` utility, see Chapter 8, “Logging Signal Data with FTP and File System Objects”.

## Signal Tracing with xPC Target Scope Blocks

Use host scopes to log signal data triggered by an event while your target application is running. This topic describes how to use the three scope block types.

---

**Note** xPC Target supports ten target scopes. It can support an infinite number of host scopes, as long as the target computer resources can support them. It can support eight file scopes. Each target scope can contain up to 10 signals. Each file or host scope can contain an infinite number of signals, as long as the target computer resources can support them.

---

---

**Note** If your model has the output of a Mux block connected to the input of an xPC Target Scope block, the signal might not be observable. To observe the signal, add a unity gain block (a Gain block with a gain of 1) between the Mux block and the xPC Target Scope block.

---

### Using xPC Target Scope Blocks from Referenced Models

You cannot add any type of xPC Target scope to a referenced model. Doing so causes an error. You can add only an xPC Target scope to the topmost model. If you want to log signals from referenced models, you can do so with the logging mechanism in xPC Target Explorer or with the xPC Target scope objects.

### Host Scope

For a host scope, the scope acquires the first N samples into a buffer. You can retrieve this buffer into the scope object property `sc.Data`. The scope then stops and waits for you to manually restart the scope.

The number of samples N to log after triggering an event is equal to the value you entered in the **Number of Samples** parameter.

Select the type of event in the Block Parameters: Scope (xPC Target) dialog box by setting **Trigger Mode** to Signal Triggering, Software Triggering, or Scope Triggering.

### Target Scope

For a target scope, logged data (`sc.Data` and `sc.Time`) is not accessible over the command-line interface on the host computer. This is because the scope object status (`sc.Status`) is never set to `Finished`. Once the scope completes one data cycle (time to collect the number of samples), the scope engine automatically restarts the scope.

If you create a scope object, for example, `sc = getscope(tg,1)` for a target scope, and then try to get the logged data by typing `sc.Data`, you get an error message:

```
Scope # 1 is of type 'Target'! Property Data is not accessible.
```

If you want the same data for the same signals on the host computer while the data is displayed on the target computer, you need to define a second scope object with type `host`. Then you need to synchronize the acquisitions of the two scope objects by setting **TriggerMode** for the second scope to `'Scope'`.

### **File Scope**

For a file scope, the scope acquires data and writes it to the file named in the **FileName** parameter in blocks of size **WriteSize**. The scope acquires the first `N` samples into the memory buffer. This memory buffer is of length **Number of Samples**. The memory buffer writes data to the file in **WriteSize** chunks. If the **AutoRestart** check box is selected, the scope then starts over again, overwriting the memory buffer. The additional data is appended to the end of the existing file. If the **AutoRestart** box is not selected, the scope collects data only up to the number of samples, and then stops. The number of samples `N` to log after triggering an event is equal to the value you entered in the **Number of Samples** parameter. If you stop, then start the scope again, the data in the file is overwritten with the new data.

Select the type of event in the Block Parameters: Scope (xPC Target) dialog box by setting **Trigger Mode** to `Signal Triggering`, `Software Triggering`, or `Scope Triggering`.

### **Signal Tracing with Simulink External Mode**

You can use Simulink external mode to establish a communication channel between your Simulink block diagram and your target application. The block diagram becomes a graphical user interface to your target application. Simulink scopes can acquire signal data from the target application, including from models referenced inside a top model.

For each Simulink scope, the xPC Target software adds a host scope to the system to upload signals. You can control which signals to upload through the External Signal & Triggering dialog box (see “Signal Selection” in the *Simulink Coder User’s Guide*).

---

**Note** Do not use Simulink external mode while xPC Target Explorer is running. Use only one interface or the other.

---

### Limitations

The following are limitations of uploading xPC Target signals to Simulink external mode:

- When setting up signal triggering (Source set to signal), you must explicitly specify the element number of the signal in the **Trigger signal:Element** field. If the signal is a scalar, enter a value of 1. If the signal is a wide signal, enter a value from 1 to 10. Do not enter Last or Any in this field when uploading xPC Target signals to Simulink scopes.
- The **Direction:Holdoff** field has no effect for the xPC Target signal uploading feature.
- Attempting to upload information from buses and virtual signals inside a reference model generates a warning.

### Before You Start

The procedures in this topic use the Simulink model `xpcosc.mdl`, which already contains a Simulink Scope block, as an example. After you download your target application to the target computer, you can connect your Simulink model to the target application.

### Signal Tracing with External Mode Example

This procedure assumes that you have downloaded your target application to the target computer.

Note that this procedure edits the Simulink window External Mode Control Panel and assumes that you are familiar with that dialog box. See “External Mode Control Panel” in the *Simulink Coder User’s Guide* for details of the Simulink external mode dialog box.

- 1 In the MATLAB window, type

```
xpcosc
```

- 2 In the Simulink window, and from the **Tools** menu, select **External Mode Control Panel**.

The **External Mode Control Panel** dialog box opens.

**3** Click the **Signal & Triggering** button.

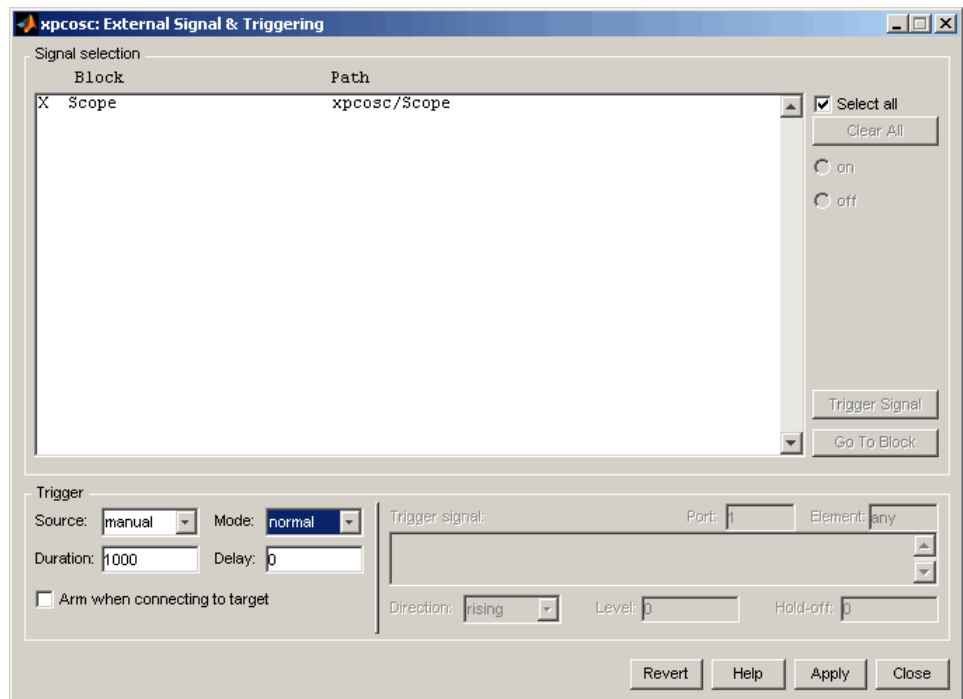
The External Signal & Triggering dialog box opens.

**4** Set the **Source** parameter to **manual**.

**5** Set the **Mode** parameter to **normal**. In this mode, the scope acquires data continuously.

**6** Select the **Arm when connecting to target** check box.

**7** In the **Duration** box, enter the number of samples for which external mode is to log data. The **External Signal & Triggering** dialog box should look similar to the figure shown.



**8** Click **Apply**, then **Close**.

- 9 In the Simulink window, increase the simulation stop time. For example, enter

50

- 10 From the **File** menu, select **Save As** and enter a filename. For example, enter `my_xpc_osc6.mdl` and then click **OK**.
- 11 Build and download the target application. In the Simulink window and from the **Tools** menu, select **Code Generation**. From the code generation submenu, select **Build Model**.

The xPC Target software downloads the target application to the default target computer.

- 12 In the Simulink window, and from the **Simulation** menu, select **External**. A check mark appears next to the menu item **External**, and Simulink external mode is activated.
- 13 If a Scope window is not displayed for the Scope block, double-click the Scope block.

A **Scope** window is displayed.

- 14 In the Simulink window, and from the **Simulation** menu, select **Connect to target**.
- 15 From the **Simulation** menu, select **Start Real-Time Code**.

The target application begins running on the target computer and the Scope window displays plotted data.



To create a host scope, use the drop-down list next to the **Add Scope** button to select **Host**. This item is set to **Target** by default.

**3** Click the **Edit** button.

The scope editing pane opens. From this pane, 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 the scope is running, the Web interface stops the scope automatically and then restarts it when the changes are made. It does not restart the scope if the state was originally stopped.

When a host scope is stopped (**Scope State** is set to **Interrupted**) or finishes one cycle of acquisition (**Scope State** is set to **Finished**), a button called **Get Data** appears on the page. If you click this button, the scope data is retrieved in comma-separated value (CSV) format. The signals in the scope are spread across columns, and each row corresponds to one sample of acquisition. The first column always corresponds to the time each sample was acquired.

---

**Note** If **Scope State** is set to **Interrupted**, the scope was stopped before it could complete a full cycle of acquisition. Even in this case, the number of rows in the CSV data will correspond to a full cycle. The last few rows (for which data was not acquired) will be set to 0.

---



# Signal Logging

**In this section...**

“Introduction” on page 5-57

“Signal Logging with xPC Target Explorer” on page 5-57

“Signal Logging in the MATLAB Interface” on page 5-60

“Signal Logging with a Web Browser” on page 5-64

## Introduction

Signal logging is the process for acquiring signal data during a real-time run, stopping the target application, and then transferring the data to the host computer for analysis. This is also known as real-time data streaming to the target computer. You can plot and analyze the data, and later save it to a disk. xPC Target signal logging samples at the base sample time. If you have a model with multiple sample rates, add xPC Target scopes to the model to sample signals at the required sample rates.

---

**Note** The xPC Target software does not support logging data with decimation.

---

---

**Note** xPC Target Explorer works with multidimensional signals in column-major format.

---

## Signal Logging with xPC Target Explorer

You plot the outputs and states of your target application to observe the behavior of your model, or to determine the behavior when you vary the input signals and model parameters.

This procedure uses a model named `xpc_osc4.mdl` as an example and assumes you have created a target application and downloaded it to the target computer. The `xpc_osc4.mdl` is the same as the `my_xpc_osc3.mdl` tutorial model with the xPC Target Scope block removed.

To create `xpc_osc4`:

- 1 In the MATLAB window, type

```
xpc_osc3
```

The `xpc_osc3` model opens.

- 2 In the Simulink window, select and delete the xPC Target Scope block and its connecting line.
- 3 From the **File** menu, click **Save as**. Enter `xpc_osc4` and then click **Save**.

You can now build and download the model (see “Building and Downloading Target Application” in the *xPC Target Getting Started Guide*).

---

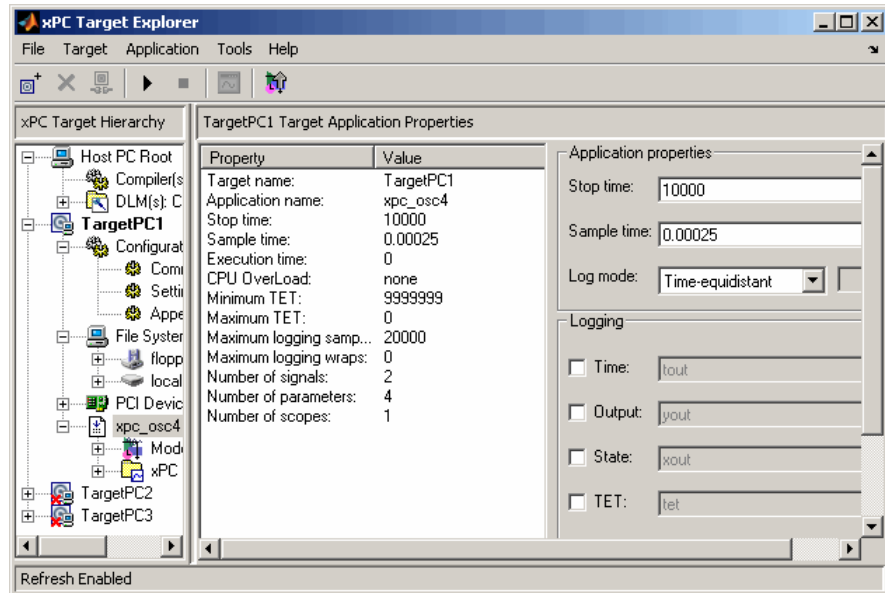
**Note** To use the xPC Target Explorer for signal logging, you need to add an Outport block to your Simulink model, and you need to activate logging on the **Data Import/Export** pane in the Configuration Parameters dialog box.

---

- 1 In xPC Target Explorer, select the downloaded target application node. For example, `xpc_osc4`.

The right pane displays the target application properties dialog box for `xpc_osc4`.

- 2 In the **Logging** pane, select the boxes of the signals you are interested in logging. For example, select **Output** and **TET**. Click **Apply**.



- 3** Start the target application. For example, in the **xPC Target Hierarchy** pane, right-click the xpc\_osc4 target application, then select **Start**.
- 4** Stop the target application. For example, in the **Target Hierarchy** pane, right-click the xpc\_osc4 target application, then select **Stop**.
- 5** Send the selected logged data to the MATLAB workspace. In the target application properties dialog box for xpc\_osc4, go to the **Logging** pane and click the **Send to MATLAB Workspace** button.

In the MATLAB desktop, the **Workspace** pane displays the logged data.

Name	Value	Min
ans	\mathworks\...	
tet	<20000x1 dou...	8.32
tg	<1x1 xpctarge...	
yout	<20000x2 dou...	-1.61

You can examine and otherwise manipulate the data.

## Signal Logging in the MATLAB Interface

You plot the outputs and states of your target application to observe the behavior of your model, or to determine the behavior when you vary the input signals and model parameters.

**Time, states, and outputs** — Logging the output signals is possible only if you add Output blocks to your Simulink model before the build process, and in the **Configuration Parameters Data Import/Export** node, select the **Save to workspace** check boxes. See “Entering Parameters for the Output Block” of the *xPC Target Getting Started Guide*.

**Task execution time** — Plotting the task execution time is possible only if you select the **Log Task Execution Time** check box in the **Configuration Parameters xPC Target options** tab. This check box is selected by default. See “Adding an xPC Target Scope Block” of the *xPC Target Getting Started Guide*.

All scopes copy the last N samples from the log buffer to the target object logs (tg.TimeLog, tg.OutputLog, tg.StateLog, and tg.TETLog). The xPC Target software calculates the number of samples N for a signal as the value of **Signal logging buffer size in doubles** divided by the number of logged signals (1 time, 1 task execution time ([TET]), outputs, states).

After you run a target application, you can plot the state and output signals. This procedure uses the Simulink model `xpc_osc3.mdl` as an example, and

assumes you have created and downloaded the target application for that model. It also assumes that you have assigned `tg` to the target computer.

- 1** In the MATLAB window, type

```
tg=xpc
```

- 2** Type

```
+tg
```

or

```
tg.start
```

or

```
start(tg)
```

The target application starts and runs until it reaches the final time set in the target object property `tg.StopTime`.

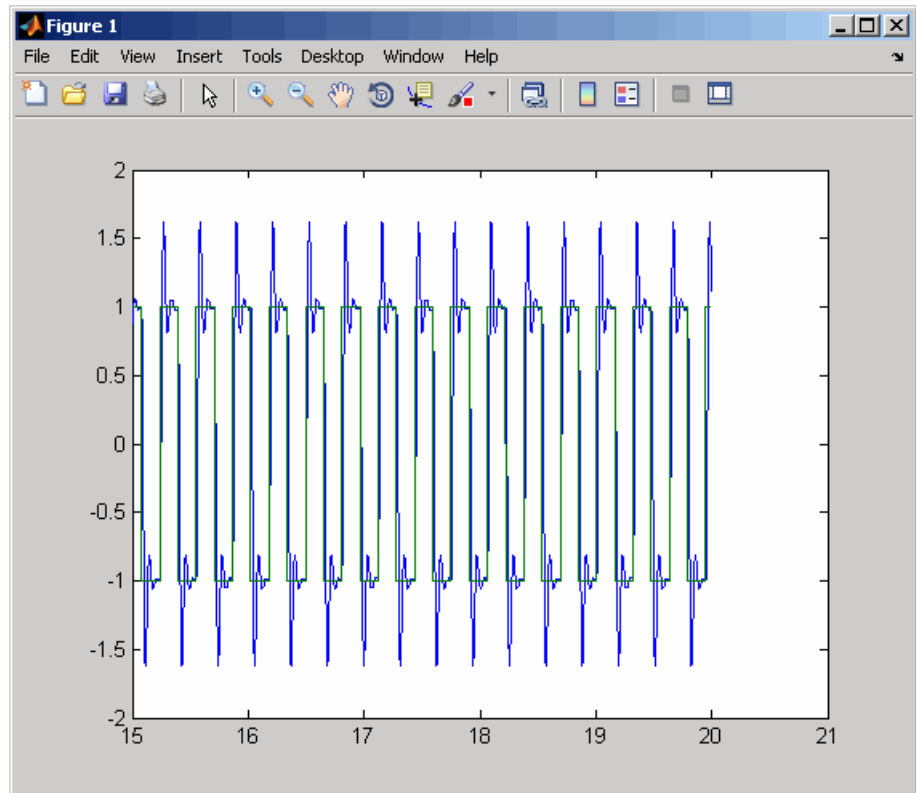
The outputs are the signals connected to Simulink Outport blocks. The model `xpcosc.mdl` has just one Outport block, labeled 1, and there are two states. This Outport block shows the signals leaving the blocks labeled `Integrator1` and `Signal Generator`.

- 3** Plot the signals from the Outport block and the states. In the MATLAB window, type

```
plot(tg.TimeLog,tg.Outputlog)
```

Values for the logs are uploaded to the host computer from the target application on the target computer. If you want to upload part of the logs, see the target object method `xpctarget.xpc.getlog`.

The plot shown below is the result of a real-time execution. To compare this plot with a plot for a non-real-time simulation, see “Simulating in Non-Real Time Using MATLAB Language” of the *xPC Target Getting Started Guide*.

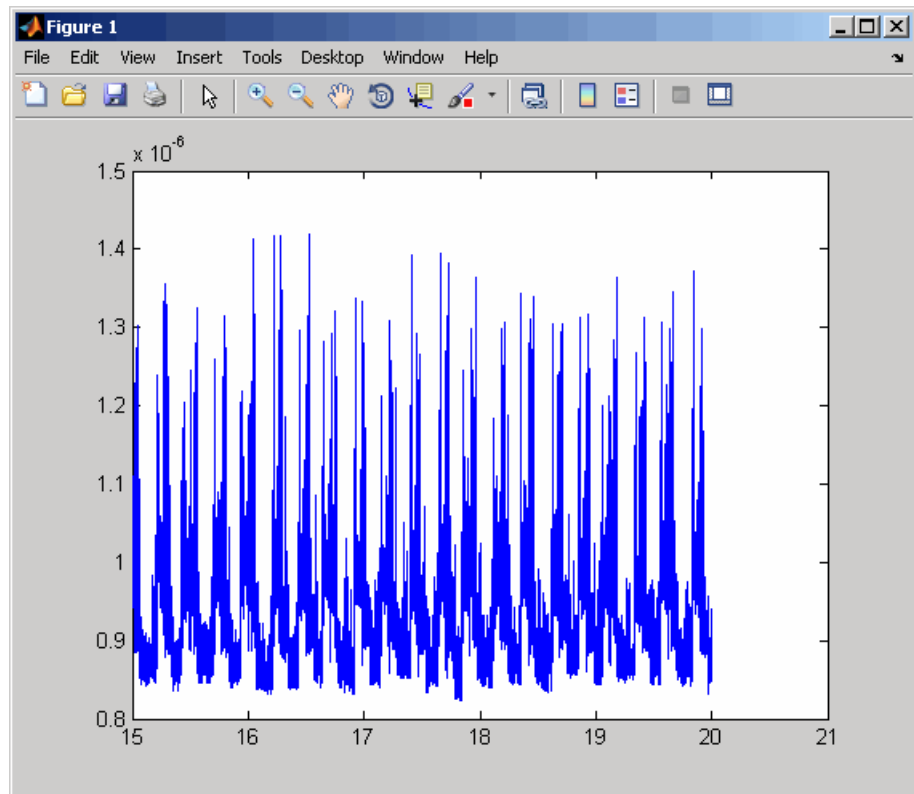


**4** In the MATLAB window, type

```
plot(tg.TimeLog,tg.TETLog)
```

Values for the task execution time (TET) log are uploaded to the host computer from the target computer. If you want to upload part of the logs, see the target object method `xpctarget.xpc.getlog`.

The plot shown below is the result of a real-time run.



The TET is the time to calculate the signal values for the model during each sample interval. If you have subsystems that run only under certain circumstances, plotting the TET would show when subsystems were executed and the additional CPU time required for those executions.

**5** In the MATLAB window, type either

```
tg.AvgTET
```

or

```
get(tg, 'AvgTET')
```

The MATLAB interface displays the following information about the average task execution time.

```
ans =  
    5.7528e-006
```

The percentage of CPU performance is the average TET divided by the sample time.

Note that each output has an associated column vector in `tg.OutputLog`. You can access the data that corresponds to a particular output by specifying the column vector for that output. For example, to access the data that corresponds to Output 2, use `tg.outputlog(:,2)`.

### Signal Logging with a Web Browser

When you stop the model execution, another section of the Web browser interface appears that enables you to download logging data. This data is in comma-separated value (CSV) format. This format can be read by most spreadsheet programs and also by the MATLAB interface using the `dlmread` function.

This section of the Web browser interface appears only if you have enabled data logging, and buttons appear only for those logs (states, output, and TET) that are enabled. If time logging is enabled, the first column of the CSV file is the time at which data (states, output, and TET values) was acquired. If time logging is not enabled, only the data is in the CSV file, without time information.

You analyze and plot the outputs and states of your target application to observe the behavior of your model, or to determine the behavior when you vary the input signals.

Time, states, and outputs — Logging the output signals is possible only if you add Outputport blocks to your Simulink model before the build process, and in the **Configuration Parameters Data Import/Export** node, select the **Save to workspace** check boxes. See “Entering Parameters for the Output Block” in *xPC Target Getting Started Guide*.



Task execution time — Logging the task execution time is possible only if you select the **Log Task Execution Time** check box in the **Configuration Parameters xPC Target options** node. This check box is selected by default. See “Entering Parameters for an xPC Target Scope Block” in *xPC Target Getting Started Guide*.

## Parameter Tuning and Inlining Parameters

In this section...
“Introduction” on page 5-66
“Parameter Tuning with xPC Target Explorer” on page 5-66
“Parameter Tuning with the MATLAB Interface” on page 5-70
“Parameter Tuning with Simulink External Mode” on page 5-73
“Parameter Tuning with a Web Browser” on page 5-75
“Saving and Reloading Application Parameters with the MATLAB Interface” on page 5-76
“Inlined Parameters” on page 5-79

### Introduction

By default, the xPC Target software lets you change parameters in your target application while it is running in real time.

---

**Note** xPC Target Explorer works with multidimensional signals in column-major format.

---

You can also improve overall efficiency by inlining parameters. The xPC Target product supports the Simulink Coder inline parameters functionality (see the Simulink Coder documentation for further details on inlined parameters). By default, this functionality makes all parameters nontunable. If you want to make some of the inlined parameters tunable, you can do so through the Model Parameter Configuration dialog box (see “Inlined Parameters” on page 5-79).

### Parameter Tuning with xPC Target Explorer

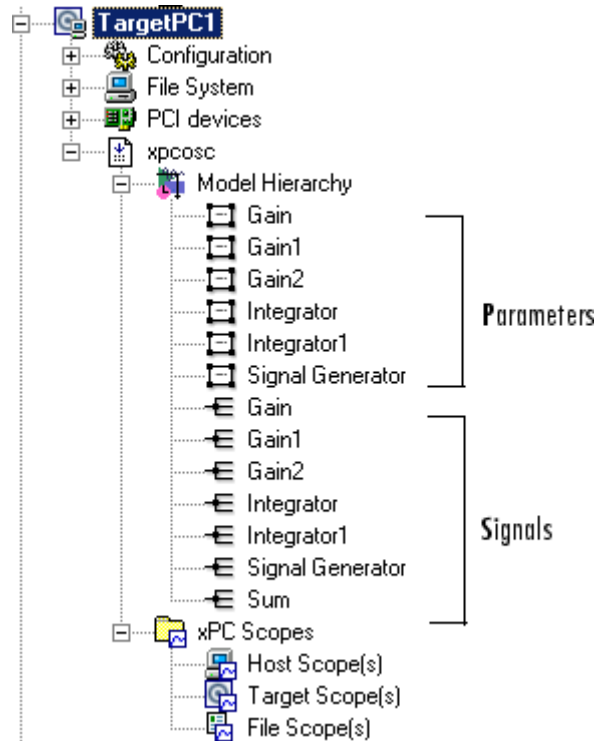
The xPC Target software lets you change parameters in your target application while it is running in real time. With these functions, you do not need to set the Simulink interface to external mode, and you do not need to connect the Simulink interface with the target application.

You can download parameters to the target application while it is running or between runs. This feature lets you change parameters in your target application without rebuilding the Simulink model. You cannot use xPC Target Explorer to change tunable source block parameters while a simulation is running.

After you download a target application to the target computer, you can change block parameters using xPC Target Explorer. This procedure uses the Simulink model `xpcosc.mdl` as an example, and assumes you have created and downloaded the target application for that model.

- 1** In xPC Target Explorer, right-click the downloaded target application node. For example, `xpcosc`.
- 2** Select **Start**.
- 3** To get the list of parameters in the target application, expand the **Model Hierarchy** node under the target application.

The Model Hierarchy expands to show the elements in the Simulink model.



The model hierarchy shows only those blocks that have tunable parameters.

- 4 Select the parameter of the signal you want to edit. For example, select Gain.

The right pane displays the block parameters dialog box for Gain. There is one parameter, **Gain**, for this block. The current value of the **Gain** parameter is displayed.

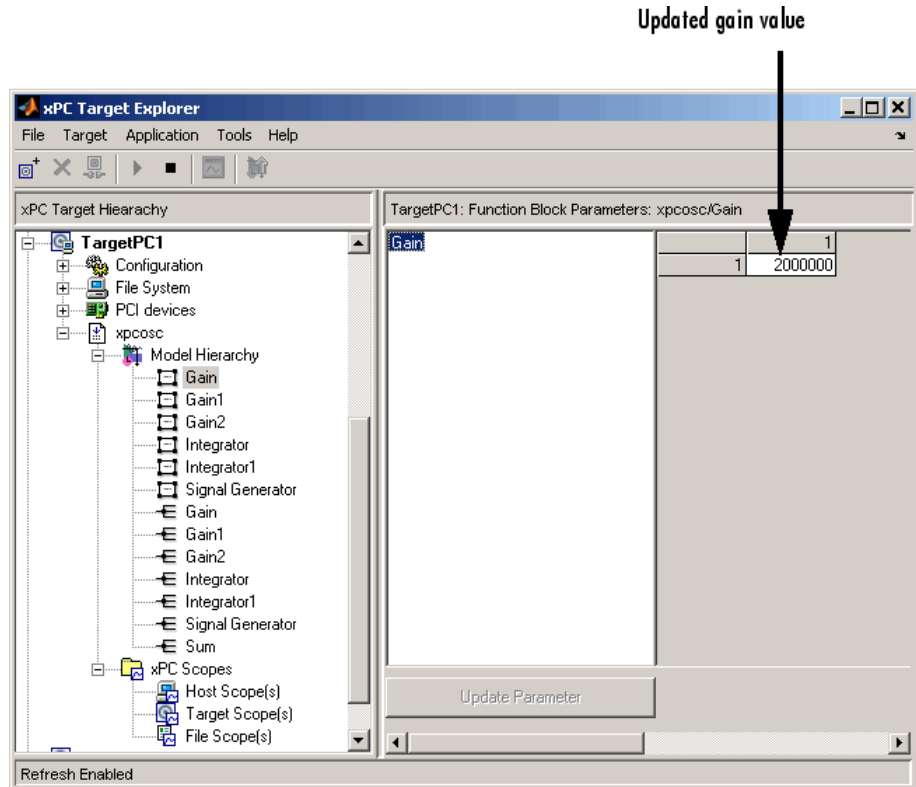
- 5 Double-click the box that contains the gain value.

The box becomes editable.

- 6 Enter a new value for the gain.

7 Press the **Enter** key.

The box is updated and the **Update Parameter** button becomes active.



The target application runs with the new parameter value and the plot frame updates the signals in any active scopes.

8 Stop the target application. For example, to stop the target application xpcosc, right-click it and select **Stop**.

The target application on the target computer stops running.

## Parameter Tuning with the MATLAB Interface

You use the MATLAB functions to change block parameters. With these functions, you do not need to set the Simulink interface to external mode, and you do not need to connect the Simulink interface with the target application.

You can download parameters to the target application while it is running or between runs. This feature lets you change parameters in your target application without rebuilding the Simulink model.

After you download a target application to the target computer, you can change block parameters using xPC Target functions. This procedure uses the Simulink model `xpcosc.mdl` as an example, and assumes you have created and downloaded the target application for that model. It also assumes that you have assigned `tg` to the target computer.

**1** In the MATLAB window, type

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

The target computer displays the following message:

```
System: execution started (sample time: 0.001000)
```

**2** Display a list of parameters. Type either

```
set(tg, 'ShowParameters', 'on')  
  
or  
  
tg.ShowParameters='on'
```

The latter command displays a list of properties for the target object.

```
ShowParameters = on
```

Parameters =

INDEX	VALUE	TYPE	SIZE	PARAMETER NAME	BLOCK NAME
0	1000000	DOUBLE	Scalar	Gain	Gain
1	400	DOUBLE	Scalar	Gain	Gain1
2	1000000	DOUBLE	Scalar	Gain	Gain2
3	0	DOUBLE	Scalar	Initial Condition	Integrator
4	0	DOUBLE	Scalar	Initial Condition	Integrator1
5	4	DOUBLE	Scalar	Amplitude	Signal Generator
6	20	DOUBLE	Scalar	Frequency	Signal Generator

- 3** Change the gain. For example, to change the Gain1 block, type either

```
tg.setparam(1,800)
```

or

```
setparam(tg,1,800)
```

As soon as you change parameters, the changed parameters in the target object are downloaded to the target application. The host computer displays the following message:

```
ans =
parIndexVec: 1
OldValues: 400
NewValues: 800
```

The target application runs and the plot frame updates the signals for any active scopes.

- 4** Stop the target application. In the MATLAB window, type

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

The target application on the target computer stops running, and the target computer displays messages like the following:

```
minimal TET: 0.000023 at time 1313.789000  
maximal TET: 0.000034 at time 407.956000
```

---

**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.

---

### Resetting Target Application Parameters to Previous Values

You can reset parameters to preceding target object property values by using xPC Target methods on the host computer. The `setparam` method returns a structure that stores the parameter index, the previous value, and the new value. If you expect to want to reset parameter values, set the `setparam` method to a variable. This variable points to a structure that stores the parameter index and the old and new parameter values for it.

**1** In the MATLAB window, type

```
pt=tg.setparam(1,800)
```

The `setparam` method returns a result like

```
pt =  
parIndexVec: 1  
OldValues: 400  
NewValues: 800
```



- 2 To reset to the previous values, type

```
setparam(tg,pt.parIndexVec,pt.OldValues)
ans =
parIndexVec: 5
OldValues: 800
NewValues: 100
```

## Parameter Tuning with Simulink External Mode

You use Simulink external mode to connect your Simulink block diagram to your target application. The block diagram becomes a graphical user interface to your target application. You set up the Simulink interface in external mode to establish a communication channel between your Simulink block window and your target application.

In Simulink external mode, wherever you change parameters in the Simulink block diagram, the Simulink software downloads those parameters to the target application while it is running. This feature lets you change parameters in your program without rebuilding the Simulink model to create a new target application.

After you download your target application to the target computer, you can connect your Simulink model to the target application. This procedure uses the Simulink model `xpcosc.mdl` as an example, and assumes you have created and downloaded the target application for that model.

- 1 In the Simulink window, and from the **Simulation** menu, click **External**.

A check mark appears next to the menu item **External**, and Simulink external mode is activated.

- 2 In the Simulink block window, and from the **Simulation** menu, click **Connect to target**.

All of the current Simulink model parameters are downloaded from the host computer to your target application.

- 3 From the **Simulation** menu, click **Start Real-Time Code**, or, in the MATLAB window, type

```
+tg
```

or

```
tg.start
```

or

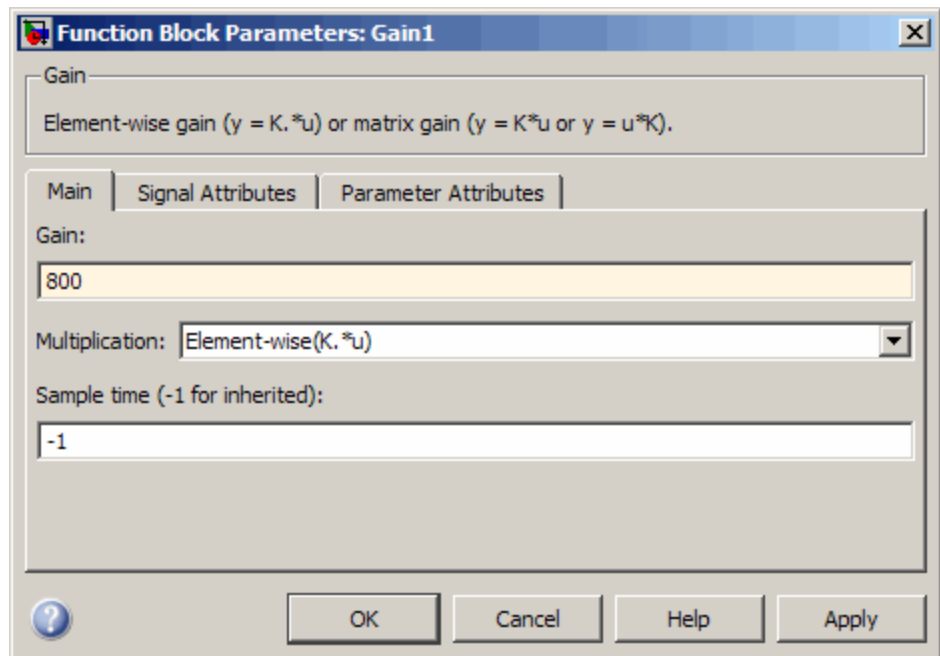
```
start(tg)
```

The target application begins running on the target computer, and the target computer displays the following message:

```
System: execution started (sample time: 0.000250)
```

- 4 From the Simulation block diagram, double-click the block labeled **Gain1**.

The Block Parameters: Gain1 parameter dialog box opens.



- 5 In the **Gain** text box, enter 800 and click **OK**.

As soon as you change a model parameter and click **OK**, or you click the **Apply** button on the Block Parameters: Gain1 dialog box, all the changed parameters in the model are downloaded to the target application.

**6** From the **Simulation** menu, click **Disconnect from Target**.

The Simulink model is disconnected from the target application. Now, if you change a block parameter in the Simulink model, there is no effect on the target application. Connecting and disconnecting the Simulink interface works regardless of whether the target application is running or not.

**7** In the MATLAB window, type either

```
stop(tg)
```

or

```
-tg
```

The target application on the target computer stops running, and the target computer displays the following messages:

```
minimal TET: 0.000023 at time 1313.789000  
maximal TET: 0.000034 at time 407.956000
```

## Parameter Tuning with a Web Browser

The **Parameters** pane 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 computer, 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 browser loads the **Parameter List** pane into the right frame.

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

If the parameter is a vector or matrix, there is a button that takes you to another page that displays the vector or matrix 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.

### **Saving and Reloading Application Parameters with the MATLAB Interface**

After you have a set of target application parameter values that you are satisfied with, you can save those values to a file on the target computer. You can then later reload these saved parameter values to the same target application. You can save parameters from your target application while the target application is running or between runs. This feature lets you save and restore parameters in your target application without rebuilding the Simulink model. You save and restore parameters with the target object methods `saveparamset` and `loadparamset`.

The procedures assume that

- You have a target application object named `tg`.
- You have assigned `tg` to the target computer.
- You have downloaded a target application to the target computer.
- You have parameters you would like to save for reuse. See
  - “Parameter Tuning with the MATLAB Interface” on page 5-70
  - “Parameter Tuning with Simulink External Mode” on page 5-73
  - “Parameter Tuning with a Web Browser” on page 5-75

#### **Saving the Current Set of Target Application Parameters**

To save a set of parameters to a target application, use the `saveparamset` method. The target application can be stopped or running.

- 1 Identify the set of parameter values you want to save.

- 2 Select a descriptive filename to contain these values. For example, use the model name in the filename. You can only load parameter values to the same target application from which you saved the parameter values.
- 3 In the MATLAB window, type either

```
tg.saveparamset('xpc_osc4_param1')
```

or

```
saveparamset(tg,'xpc_osc4_param1')
```

The xPC Target software creates a file named `xpcosc4_param1` in the current folder of the target computer, for example, `C:\xpcosc4_param1`.

For a description of how to restore parameter values to a target application, see “Loading Saved Parameters to a Target Application” on page 5-77. For a description of how to list the parameters and values stored in the parameter file, see “Listing the Values of the Parameters Stored in a File” on page 5-78.

## Loading Saved Parameters to a Target Application

To load a set of saved parameters to a target application, use the `loadparamset` method. You must load parameters to the same model from which you save the parameter file. If you load a parameter file to a different model, the behavior is undefined.

This section assumes that you have a parameters file saved from an earlier run of `saveparamset` (see “Saving the Current Set of Target Application Parameters” on page 5-76).

- 1 From the collection of parameter value files on the target computer, select the one that contains the parameter values you want to load.
- 2 In the MATLAB window, type either

```
tg.loadparamset('xpc_osc4_param1')
```

or

```
loadparamset(tg,'xpc_osc4_param1')
```

The xPC Target software loads the parameter values into the target application.

---

**Tip** To enable the software to load the parameter set automatically during startup, see “Load a parameter set from a file on the designated target file system”.

---

For a description of how to list the parameters and values stored in the parameter file, see “Listing the Values of the Parameters Stored in a File” on page 5-78.

### Listing the Values of the Parameters Stored in a File

To list the parameters and their values, load the file for a target application, then turn on the ShowParameters target object property.

This section assumes that you have a parameters file saved from an earlier run of saveparamset (see “Saving the Current Set of Target Application Parameters” on page 5-76).

**1** Stop the target application. For example, type

```
tg.stop
```

**2** Load the parameter file. For example, type

```
tg.loadparamset('xpc_osc4_param1');
```

**3** Display a list of parameters. For example, type

```
tg.ShowParameters='on';
```

and then type

```
tg
```

The MATLAB window displays a list of parameters and their values for the target object.

## Inlined Parameters

This procedure describes how you can globally inline parameters for a model, then specify which of these parameters you still want to be tunable. It assumes that you are familiar with how to build target applications (if you are not, read the *xPC Target Getting Started Guide* first). After you have performed this procedure, you will be able to tune these parameters.

- “Tuning Inlined Parameters with xPC Target Explorer” on page 5-81
- “Tuning Inlined Parameters with the MATLAB Interface” on page 5-84

---

**Note** You cannot tune inlined parameters that are structures.

---

The following procedure uses the Simulink model `xpcosc.mdl` as an example.

- 1** In the MATLAB Command Window, type

```
xpcosc
```

The model is displayed in the Simulink window.

- 2** Select the blocks of the parameters you want to make tunable. For example, this procedure makes the signal generator’s amplitude parameter tunable. Use the variable `A` to represent the amplitude.
- 3** Double-click the Signal Generator block and enter `A` for the Amplitude parameter. Click **OK**.
- 4** In the MATLAB Command Window, assign a constant to that variable. For example, type

```
A = 4
```

The value is displayed in the MATLAB workspace.

- 5** In the Simulink window, from the **Simulation** menu, click **Model Configuration Parameters**.

The Configuration Parameters dialog box for the model is displayed.

**6** Select the **Signals and Parameters** node under **Optimization**.

**7** In the rightmost pane, select the **Inline parameters** check box.

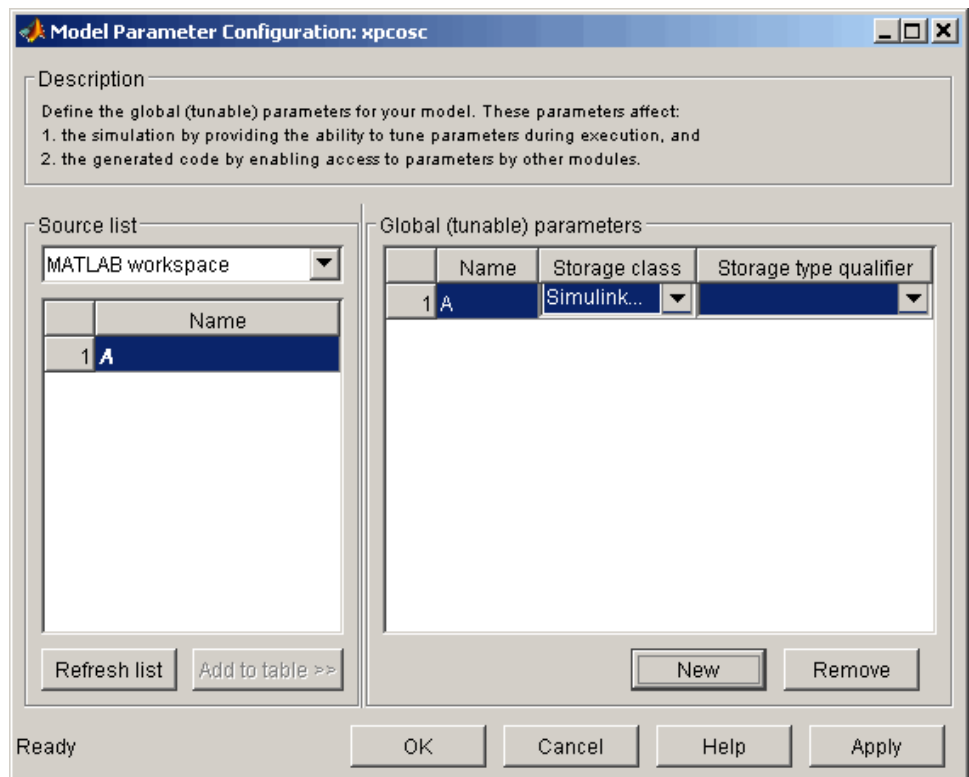
The **Configure** button is enabled.

**8** Click the **Configure** button.

The Model Parameter Configuration dialog box is displayed. Note that the MATLAB workspace contains the constant you assigned to A.

**9** Select the line that contains your constant and click **Add to table**.

The Model Parameter Configuration dialog box appears as follows.





If you have more global parameters you want to be able to tune, add them also.

- 10** Click **Apply**, then click **OK**.
- 11** In the Configuration Parameters dialog, click **Apply**, then **OK**.
- 12** If you want, increase the model stop time, or set it to `inf`.
- 13** When you are finished, click **Apply**, then **OK**, and save the model. For example, save it as `xpc_osc5.mdl`.
- 14** Build and download the model to your target computer.

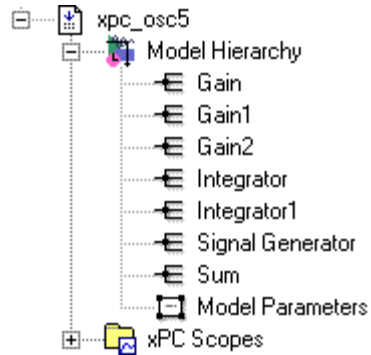
You next can use xPC Target Explorer or the MATLAB interface to work with the tunable parameters.

### **Tuning Inlined Parameters with xPC Target Explorer**

This procedure describes how you can tune inlined parameters through the xPC Target Explorer. It assumes that you have built and downloaded the model from the topic “Inlined Parameters” on page 5-79 to the target computer. It also assumes that the model is running.

- 1** If you have not yet started xPC Target Explorer, do so now. Be sure it is connected to the target computer to which you downloaded the `xpc_osc5` target application.

- 2 To get the list of tunable inlined parameters in the target application, expand the target application node, then expand the Model Hierarchy node under the target application node.



Note that the Model Hierarchy node displays a list of signals and an object called Model Parameters. Model Parameters contains the list of tunable inlined parameters.

- 3 To display the tunable parameters, select Model Parameters.

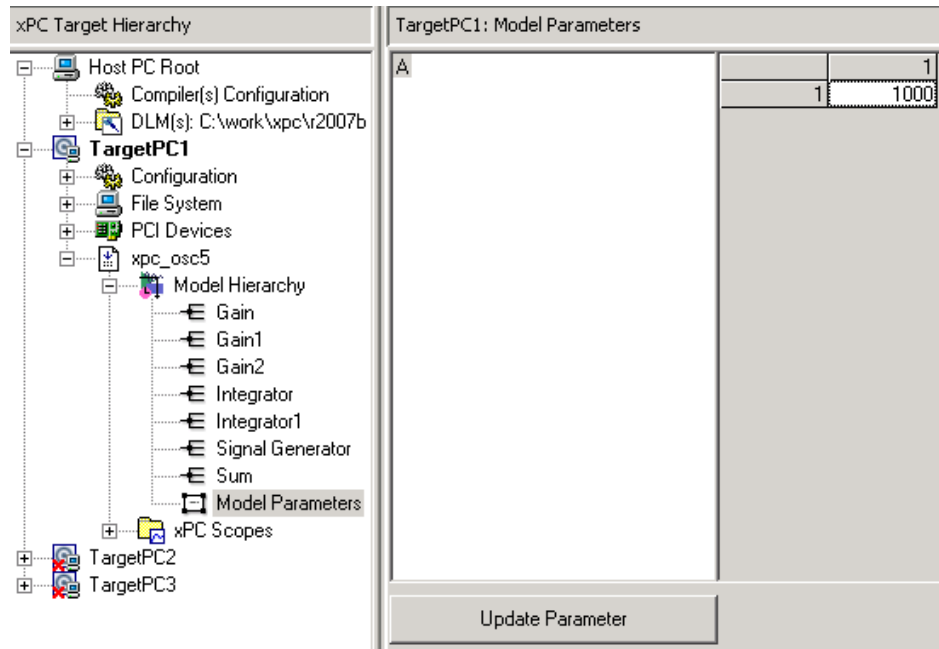
The constant A and its value are shown in the right pane.

- 4 Double-click the box that contains the tunable parameter A.

The box becomes editable.

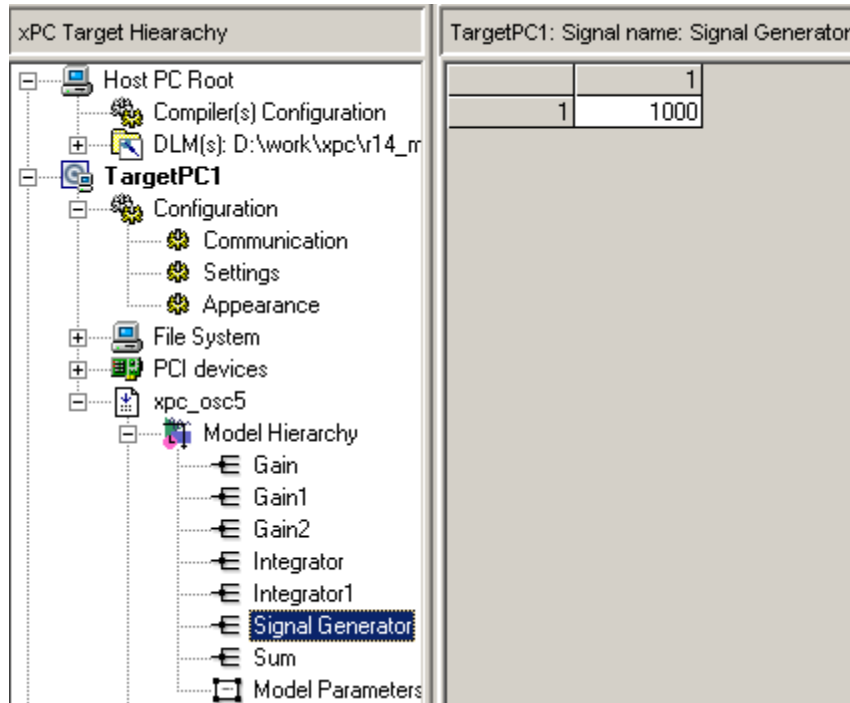
- 5 Enter a new value for the parameter and press **Enter**.

The box is updated and the **Update Parameter** button becomes active.



- 6** To apply the new value, click the **Update Parameter** button.
- 7** To verify the updated value, select the signal object associated with A. For example, select **Signal Generator**.

The value of Signal Generator is shown in the right pane.



8 Stop the target application.

### Tuning Inlined Parameters with the MATLAB Interface

This procedure describes how you can tune inlined parameters through the MATLAB interface. It assumes that you have built and downloaded the model from the topic “Inlined Parameters” on page 5-79 to the target computer. It also assumes that the model is running.

You can tune inlined parameters using a parameter ID as you would conventional parameters.

- Use the `getparamid` function to get the ID of the inlined parameter you want to tune. For the `block_name` parameter, leave a blank ( ' ').
- Use the `setparam` function to set the new value for the inlined parameter.

- 1 Save the following code in a MATLAB file. For example, `change_inlineA`.

```

tg=xpc; %Create xPC Target object
pid=tg.getparamid('','A'); %Get parameter ID of A
if isempty(pid) %Check value of pid.
    error('Could not find A');
end
tg.setparam(pid,100); %If pid is valid, set parameter value.

```

- 2 Execute that MATLAB file. For example, type

```
change_inlineA
```

- 3 To see the new parameter value, type

```
tg.showparameters='on'
```

The `tg` object information is displayed, including the parameter lines:

```
NumParameters = 1
```

```
ShowParameters = on
```

```
Parameters = INDEX  VALUE  TYPE  SIZE  PARAMETER NAME  BLOCK
NAME
```

```
0      100  DOUBLE Scalar A
```

# Nonobservable Signals and Parameters

Observable signals are those you can monitor, trace, and log. Nonobservable signals are those that exist in the target application, but are not observable from the host computer.

You cannot observe the following types of signals:

- Virtual or bus signals (including all signals from bus and virtual blocks). You can access these signals from nonvirtual source blocks.

---

### Tip

- To observe a virtual signal, add a Gain block with gain 1.0 (unit gain) and observe its output.
  - To observe a virtual bus, add a Gain block with unit gain to each individual signal.
- 

- Signals that you have optimized with block reduction optimization. You can access these signals by making them test points.
- Signals of complex or multiword data types.

Observable parameters are those you can tune. Nonobservable parameters are those that exist in the target application, but are not tunable from the host computer. You cannot observe the parameters of complex or multiword data types.

# Execution Modes

---

- “Introducing Execution Modes” on page 6-2
- “Interrupt Mode” on page 6-3
- “Polling Mode” on page 6-5

## Introducing Execution Modes

### Introduction

Interrupt mode is the default real-time execution mode for the xPC Target kernel. In certain conditions, you might want to change the real-time execution mode to polling mode. A good understanding of interrupt and polling modes will help you to use them effectively, and to decide under which circumstances it makes sense for you to switch to the polling mode.

A third execution mode, freerun, is also available. In this mode, the target application thread does not wait for the timer and the kernel runs the application as fast as possible. The time between each execution might vary if the target application has any conditional code. The three execution modes are mutually exclusive. For a description of how to use the freerun mode, see “Entering Simulation Parameters” in the *xPC Target Getting Started Guide*.



## Interrupt Mode

Interrupt mode is the default real-time execution mode for the kernel. This mode provides the greatest flexibility and is the mode you should choose for any application that executes at the given base sample time without overloading the CPU.

The scheduler implements real-time single-tasking and multitasking execution of single-rate or multirate systems, including asynchronous events (interrupts). Additionally, background tasks like host-target communication or updating the target screen run in parallel with sample-time-based model tasks. This allows you to interact with the target system while the target application is executing in real time at high sample rates. This is made possible by an interrupt-driven real-time scheduler that is responsible for executing the various tasks according to their priority. The base sample time task can interrupt any other task (larger sample time tasks or background tasks) and execution of the interrupted tasks resumes as soon as the base sample time task completes operation. This gives a quasi parallel execution scheme with consideration to the priorities of the tasks.

### Latencies Introduced by Interrupt Mode

Compared to other modes, interrupt mode has more advantages. The exception is the disadvantage of introducing a constant overhead, or latency, that reduces the minimal possible base sample time to a constant number. The overhead is the sum of various factors related to the interrupt-driven execution scheme and can be referred to as overall interrupt latency. The overall latency consists of the following parts, assuming that the currently executing task is not executing a critical section and has therefore not disabled any interrupt sources:

- Interrupt controller latency — In a PC-compatible system the interrupt controller is not part of the x86-compatible CPU but part of the CPU chip set. The controller is accessed over the I/O-port address space, which introduces a read or write latency of about 1  $\mu$ s for each 8-bit/16-bit register access. Because the CPU has to check for the interrupt line requesting an interrupt, and the controller has to be reset after the interrupt has been serviced, a latency of about 5  $\mu$ s is introduced for the interrupt controller.

- CPU hardware latency — Modern CPUs try to predict the next couple of instructions, including branches, by the use of instruction pipelines. If an interrupt occurs, the prediction fails and the pipeline has to be fully reloaded. This process introduces an additional latency. Additionally, because of interrupts, cache misses will occur.
- Interrupt handler entry and exit latency — Because an interrupt can stop the currently executing task at any instruction and the interrupted task has to resume when the interrupting task completes execution, its state has to be saved and restored accordingly. This includes saving CPU data and address registers, including the stack pointer. In the case that the interrupted task executed floating-point unit (FPU) operations, the FPU stack has to be saved as well (108 bytes on a Pentium CPU). This introduces additional latency.
- Interrupt handler content latency — If a background task has been executing for some time, say in a loop, its data will be available in the cache. When an interrupt occurs and the interrupt service handler is executed, the interrupt handler data might no longer be in the cache, causing the CPU to reload it from slower RAM. This introduces additional latency. Because of its unpredictable nature, an interrupt generally reduces the optimal execution speed or introduces latency.

The kernel in interrupt mode is close to optimal for executing code on a PC-compatible system. However, interrupt mode introduces an overall latency of about 8  $\mu$ s. This is a significant amount of time when considering that a 1 GHz CPU can execute thousands of instructions within 8  $\mu$ s. This time is equivalent to a Simulink model containing a hundred nontrivial blocks. Additionally, because lower priority tasks have to be serviced as well, at least 5% of headroom is required, which can cause additional cache misses and therefore nonoptimal execution speed.

## Polling Mode

Polling mode for the kernel is designed to execute target applications at sample times close to the limit of the hardware (CPU). Using polling mode with high-speed and low-latency I/O boards and drivers allows you to achieve smaller sample times for applications that you cannot achieve using the interrupt mode of the xPC Target software.

Polling mode has two main applications:

- Control applications — Control applications of average model size and I/O complexity that are executed at very small sample times ( $T_s = 5$  to  $50 \mu\text{s}$ )
- DSP applications — Sample-based DSP applications (mainly audio and speech) of average model size and I/O complexity that are executed at very high sample rates ( $F_s = 20$  to  $200 \text{ kHz}$ )

### Introducing Polling Mode

Polling mode for the kernel does not have the  $8 \mu\text{s}$  of latency that interrupt mode does. This is because the kernel does not allow interrupts at all, so the CPU can use this extra time for executing model code.

Polling mode is sometimes seen as a “primitive” or “brute force” real-time execution scheme. Nevertheless, when a real-time application executes at a given base sample time in interrupt mode and overloads the CPU, switching to polling mode is often the only alternative to get the application to execute at the required sample time.

*Polling* means that the kernel waits in an empty while loop until the time at which the next model step has to be executed is reached. Then the next model step is executed. At least a counter implemented in hardware has to be accessible by the kernel in order to get a base reference for when the next model step execution has to commence. The kernel polls this hardware counter. If this hardware counter must be outside the CPU, e.g., in the chip set or even on an ISA or PCI board, the counter value can only be retrieved by an I/O or memory access cycle that again introduces latency. This latency usually eats up the freed-up time of polling mode. Fortunately, since the introduction of the Pentium CPU family from Intel, the CPU is equipped with a 64 bit counter on the CPU substrate itself, which commences counting at power-up time and counts up driven by the actual clock rate of the CPU.

Even a highly clocked CPU is not likely to lead to an overflow of a 64 bit counter ( $2^{64} * 1e-9$  (1 GHz CPU) = 584 years). The Pentium counter comes with the following features:

- More precise measurements — Because the counter counts up with the CPU clock rate (~1 GHz nowadays), time measurements even in the microsecond range are very precise, leading to small real-time errors.
- Overflow handler not required— Because the counter is 64 bits wide, in practical use overflow does not occur, avoiding the CPU-time overhead of handling overflows.
- Minimal latency — The counter resides on the CPU. Reading the counter value can be done within one CPU cycle, introducing minimal latency.

The polling execution scheme does not depend on any interrupt source to notify the code to continue calculating the next model step. While this frees the CPU, it means that any code that is part of the exclusively running polling loop is executed in real time, even components, which have so far been executed in background tasks. Because these background tasks are usually non-real-time tasks and can use a lot of CPU time, do not execute them. This is the main disadvantage of polling mode. To be efficient, only the target application's relevant parts should be executed. In the case of the xPC Target software, this is the code that represents the Simulink model itself.

Therefore, host-target communication and target display updating are disabled. Because polling mode reduces the features of the xPC Target product to a minimum, you should choose it only as the last possible alternative to reach the required base sample time for a given model. Therefore, do the following before you consider polling mode:

- The model is optimal concerning execution speed — First, you should run the model through the Simulink profiler to find any possible speed optimizations using alternative blocks. If the model contains continuous states, the discretization of these states will reduce model complexity significantly, because a costly fixed-step integration algorithm can be avoided. If continuous states cannot be discretized, you should use the integration algorithm with the lowest order that still produces the required numerical results.

- Use the fastest available computer hardware — Use the CPU with the highest clock rate available for a given PC form factor. For the desktop form factor, this would mean a clock rate above 3 GHz; for a mobile application, e.g., using the PC/104 form factor, this would mean a clock rate above 1 GHz. Executing `xpcbench` at the MATLAB prompt gives a relative measure of CPU performance when running typical target applications.
- Use the lowest latency I/O hardware and drivers available — Many xPC Target applications communicate with hardware through I/O hardware over either an ISA or PCI bus. Because each register access to such I/O hardware introduces a comparably high latency time ( $\sim 1 \mu\text{s}$ ), the use of the lowest latency hardware/driver technology available is crucial.
- The base sample time is about  $50 \mu\text{s}$  or less — The time additionally assigned to model code execution in polling mode is only about  $8 \mu\text{s}$ . If the given base sample time of the target application exceeds about  $50 \mu\text{s}$ , the possible percentage gain is rather small. Other optimization technologies might have a bigger impact on increasing performance.

## Setting the Polling Mode

Polling mode is an alternative to the default interrupt mode of the kernel. This means that the kernel on the bootable media created by the GUI allows running the target application in both modes without using another boot disk.

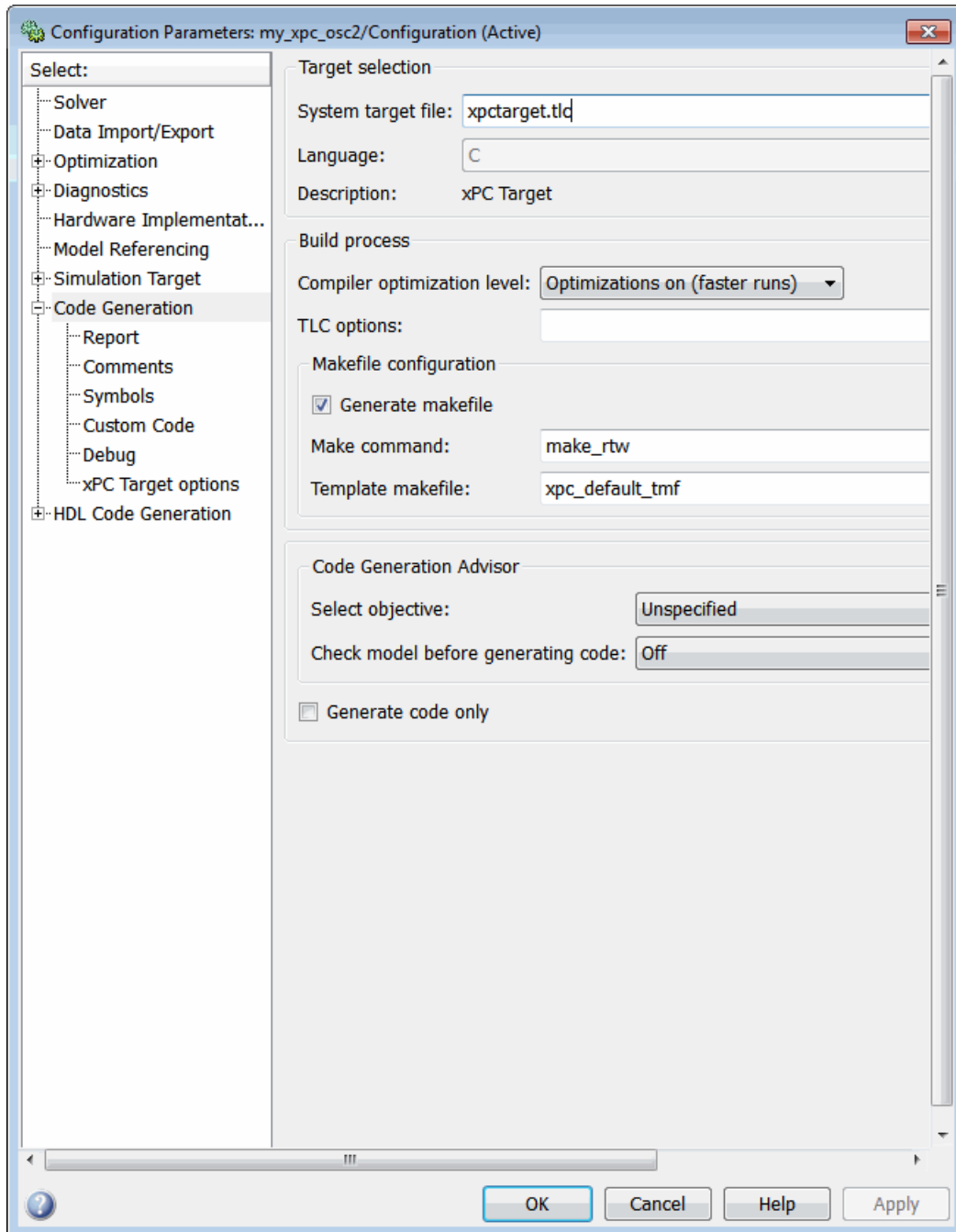
By default the target application executes in interrupt mode. To switch to polling mode, you need to pass an option to the **System target file** command.

The following example uses `xpcosc.mdl`.

- 1 In the Simulink window, and from the **Tools** menu, point to **Code Generation**, and then click **Options**.

The Configuration Parameters dialog box opens.

- 2 In the left pane, click the **Code Generation** node.



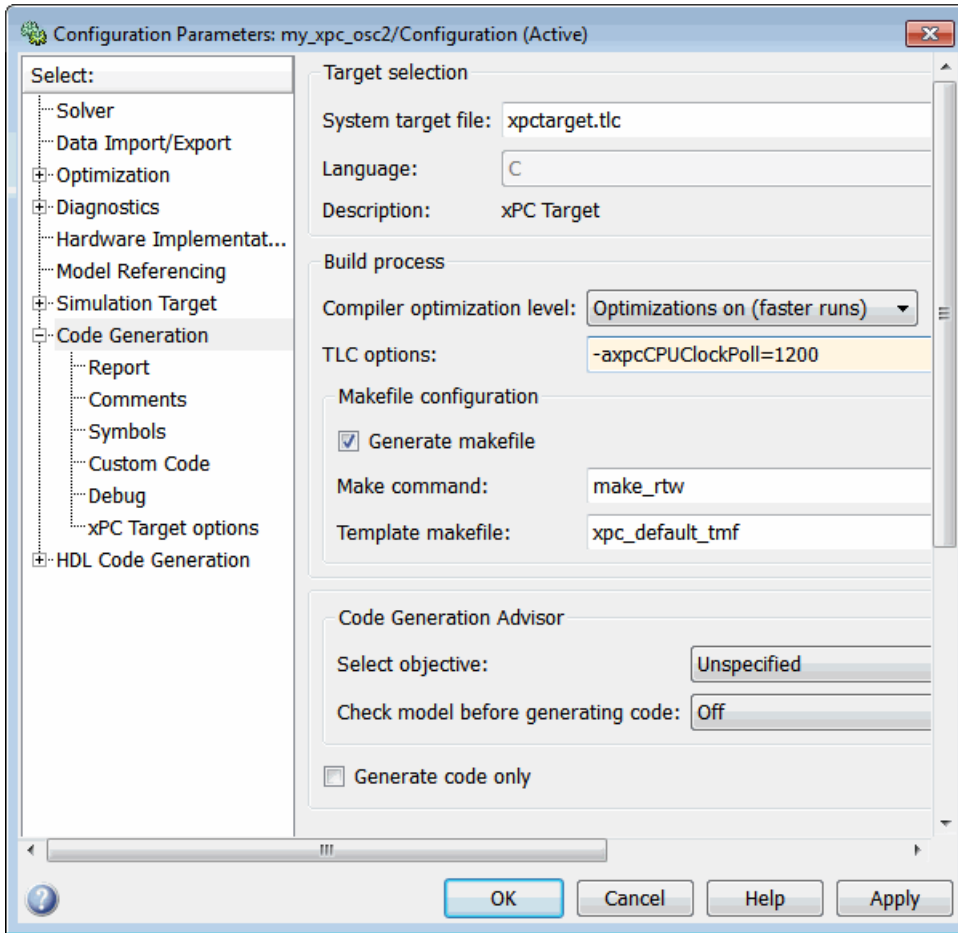
**3** In the **TLC options** edit field, specify the option

```
-axpcCPUClockPoll=CPUClockRateMHz
```

You must assign the target computer CPU clock rate because the Pentium's on-chip counter used for polling mode counts up with the CPU clock rate. If the clock rate is provided, the kernel can convert clock ticks to seconds and vice versa. If the required clock rate is not provided, the target application does not execute at the required base sample time. You can find out about the CPU clock rate of the target computer by rebooting the target computer and checking the screen output during BIOS execution time. The BIOS usually displays the CPU clock rate in MHz right after the target computer has been powered up.

For example, if your target computer is a 1.2 GHz AMD Athlon, specify the following option in the **TLC options** edit field:

```
-axpcCPUClockPoll=1200
```



If you want to execute the target application in interrupt mode again, either remove the option or assign a CPU clock rate of 0 to the option:

```
-axpcCPUClockPoll=0
```

If you make a change to the **TLC options** field, you need to rebuild the target application for the change to take effect. Building the target application, downloading it, and preparing it for a run then work exactly the same way as they did with default interrupt mode.



After the download of the target application has succeeded, the target screen displays the mode, and if polling mode is activated, it additionally displays the defined CPU clock rate in MHz. This allows you to check the setting.

## Restrictions Introduced by Polling Mode

As explained above, polling mode executes the Simulink-based target application in real time exclusively. While the target application is executing in polling mode, all background tasks, including those for host-target communication, target screen updating, and UDP transfers, are inactive. This is because all interrupts of the target computer are fully disabled during the execution of the target application. On one hand, this improves polling performance; on the other hand, background tasks are not serviced.

The following topics list all relevant restrictions of polling mode, which are otherwise available in the default interrupt mode.

### Host-Target Communication Is Not Available During the Execution of the Target Application

If the target application execution is started in polling mode, e.g., with

```
start(tg)
```

host-target communication is disabled throughout the entire run, or in other words until the stop time is reached. Each attempt to issue a command like

```
tg
```

leads to a communication-related error message. Even the `start(tg)` command to start polling mode execution returns such an error message, because the host side does not receive the acknowledgment from the target before timing out. The error message when executing `start(tg)` is not avoidable. Subsequently, during the entire run, it is best not to issue any target-related commands on the host, in order to avoid displaying the same error message over and over again.

As a consequence, it is not possible to issue a `stop(tg)` command to stop the target application execution from the host side. The target application has to reach its set stop time for polling mode to be exited. You can use

```
tg.stoptime=x
```

before starting the execution, but once started the application executes until the stop time is reached.

Nevertheless, there is a way to stop the execution interactively before reaching the target application stop time. See “Controlling the Target Application” on page 6-14.

If the target application execution finally reaches the stop time and polling mode execution is stopped, host-target communication will begin functioning again. However, the host-target communication link might be in a bad state. If you still get communication error messages after polling mode execution stops, type the command

```
xpctargetping
```

to reset the host-target communication link.

After the communication link is working again, type

```
tg
```

to resync the target object on the host side with the most current status of the target application.

### **Target Screen Does Not Update During the Execution of the Target Application**

As with the restriction mentioned above, target screen updating is disabled during the entire execution of the target application. Using the kernel with the **Enable target scope** option enabled (see `xpcexplr` GUI) does not work. You should therefore use the kernel with the **Enable target scope** property disabled (text output only). The kernel enabled with text mode actually provides more information when running in polling mode.

## Session Time Does Not Advance During the Execution of the Target Application

Because all interrupts are disabled during a run, the session time does not advance. The session time right before and after the run is therefore the same. This is a minor restriction and should not pose a problem.

## The Only Rapid-Prototyping Feature Available Is Data Logging

Because host-target communication and target screen updating are disabled during the entire run, most of the common rapid-prototyping features of the xPC Target product are not available in polling mode. These include

- Parameter tuning — Neither through the command-line interface nor through Simulink external mode
- Through scope objects — Not through the following types of scope objects:
  - host (xPC Target Explorer or scripts)
  - target (scopes on the target screen if property **Enable target scope** is enabled)
  - file (xPC Target Explorer, scripts, or blocks, on target computers that have file systems)
- Signal monitoring — You cannot run a GUI interface on the host computer using an environment that depends on communication between the host and target computers.
- Applications using the xPC Target API
- The Internet browser interface
- Other utilities like `xpctargetspy`

The only rapid-prototyping feature available is signal logging, because the acquisition of signal data runs independently from the host, and logged data is retrieved only after the execution is stopped. Nevertheless, being able to log data allows gathering good enough information about the behavior of the target application. Signal logging becomes a very important feature in polling mode.

## **Multirate Simulink Models Cannot Be Executed in Multitasking Mode on the Target Computer**

Because of the polling mode execution scheme, executing Simulink-based target applications in multitasking mode is not possible. The modeling of function-call subsystems to handle asynchronous events (interrupts) is not possible either. This can be a hard restriction, especially for multirate systems. Multirate systems can be executed in single-tasking mode, but because of its sequential execution scheme for all subsystems with different rates, the CPU will most likely overload for the given base sample time. As an important consequence, polling mode is only a feasible alternative to interrupt mode if the model has a single rate or if it can be converted to a single-rate model. A single-rate model implies continuous states only, discrete states only, or mixed continuous and discrete states, if the continuous and discrete subsystems have the same rate. Use the Simulink **Format > Sample time color** feature to check for the single rate requirement. Additionally, set the tasking mode property in the **Simulation menu Configuration Parameters > Solver** pane to `SingleTasking` to avoid a possible switch to multitasking mode. For more information on single-tasking mode compared to multitasking mode, see the *Simulink Coder User's Guide* documentation.

## **I/O Drivers Using Kernel Timing Information Cannot Be Used Within a Model**

Some xPC Target drivers use timing information exported from the kernel to, for example, detect time-outs. Because the standard timing engine of the kernel does not run in polling mode, the required timing information is not passed back to the drivers. Therefore, in polling mode you cannot use drivers that import the header file `time_xpcimport.h`. This is a current restriction only. In a future version of polling mode, all drivers will make use of the Pentium counter for getting timing information instead.

## **Controlling the Target Application**

As mentioned, there is no way to interact with the running target application in polling mode. This is especially restrictive for the case of stopping the model execution before the application has reached the stop time that was defined before the execution started. Because polling mode tries to be as optimal as possible, any rapid-prototyping feature except signal logging is disabled. But because I/O driver blocks added to the model are fully functional, you can use I/O drivers to get to a minimal level of interactivity.

Stopping a target application using polling mode — You can use a low-latency digital input driver for the digital PCI board in your model, which reads in a single digital TTL signal. The signal is TTL low unless the model execution should be stopped, for which the signal changes to TTL high. You can connect the output port of the digital input driver block to the input port of a Stop simulation block, found in the standard Simulink block library. This stops the execution of the target application, depending on the state of the digital input signal. You can either use a hardware switch connected to the board-specific input pin or you can generate the signal by other means. For example, you could use another digital I/O board in the host machine and connect the two boards (one in the host, the other in the target) over a couple of wires. You could then use the Data Acquisition Toolbox™ product to drive the corresponding TTL output pin of the host board to stop the target application execution from within the MATLAB interface.

Generally, you can use the same software/hardware setup for passing other information back and forth during run time of the target application. It is important to understand that any additional feature beside signal logging has to be implemented at the model level, and it is, therefore, the user's responsibility to optimize for the minimal additional latency the feature introduces. For example, being able to interactively stop the target application execution is paid for by the additional 1  $\mu$ s latency required to read the digital signal over the digital I/O board. However, if you need to read digital inputs from the plant hardware anyway, and not all lines are used, you get the feature for free.

## Polling Mode Performance

This is preliminary information. All benchmarks have been executed using a 1 GHz AMD Athlon machine. For more information about benchmarks, see `xpcbench` or type `help xpcbench` in the MATLAB Command Window.

The minimum achievable base sample time for benchmark model `Minimal` is 1  $\mu$ s with signal logging disabled and 2  $\mu$ s with signal logging enabled.

The minimum achievable base sample time for model `f14` using an `ode4` fixed-step integration algorithm is 4  $\mu$ s with signal logging disabled and 5  $\mu$ s with signal logging enabled.

A more realistic model, which has been benchmarked, is a second-order continuous controller accessing real hardware over two 16 bit A/D channels and two 16 bit D/A channels. The analog I/O board used is the fast and low-latency PMC-ADADIO from <http://www.generalstandards.com>, which is used in conjunction with some recently developed and heavily optimized (lowest latency) xPC Target drivers for this particular board. The minimum achievable base sample time for this model using an ode4 fixed-step integration algorithm is 11  $\mu$ s with signal logging disabled and 12  $\mu$ s with signal logging enabled. This equals a sample rate of almost 100 kHz. The achievable sample time for the same model in interrupt mode is  $\sim$ 28  $\mu$ s or a sample rate of  $\sim$ 33 kHz. For this application, the overall performance increase using polling mode is almost a factor of 3.

### **Polling Mode and Multicore Processors**

If your target computer has multicore processors, enabling the multicore processor supports removes the following restrictions. Other restrictions still apply. (For details on how to enable multicore processor support, see the `setxpcenv` function or the “Changing Environment Properties with xPC Target Explorer” on page 4-16 topic in the Chapter 4, “Target Application Environment” chapter.)

- Host-target communication is now available during the execution of the target application.
- Target screen now updates during the execution of the target application.
- External interrupts are now allowed during the execution of the real-time model. This does not mean that you can trigger your model with an external interrupt.
- File scopes can now log data into a file on the target computer.

# Execution Using MATLAB Scripts

---

An important part of the “Rapid Prototyping” and “Hardware in the Loop” workflows is preparing stress test and regression test scripts. The xPC Target product includes specialized MATLAB classes and functions for setting up the target environment, booting the target computer, loading and running the target application, and displaying and recording the results. You can do these tasks using MATLAB functions and target and scope class objects.

- Chapter 7, “Targets and Scopes in the MATLAB Interface”
- Chapter 8, “Logging Signal Data with FTP and File System Objects”





# Targets and Scopes in the MATLAB Interface

---

You can work with xPC Target target and scope objects through the MATLAB interface (MATLAB Command Window), the target computer command line, a Web browser, or an xPC Target API. This chapter describes how to use the MATLAB interface to work with the target and scope objects in the following sections. See Chapter 10, “Execution Using the Target Computer Command Line” for a description of the target computer command-line interface.

- “Target Driver Objects” on page 7-2
- “Target Scope Objects” on page 7-8

## Target Driver Objects

In this section...
“What Is a Target Object?” on page 7-2
“Accessing Help for Target Objects” on page 7-3
“Creating Target Objects” on page 7-3
“Displaying Target Object Properties” on page 7-4
“Setting Target Object Properties from the Host Computer” on page 7-5
“Getting the Value of a Target Object Property” on page 7-6
“Using the Method Syntax with Target Objects” on page 7-7

### What Is a Target Object?

The xPC Target software uses a target object (of class `xpctarget.xpc`) to represent the target kernel and your target application. Use target object functions to run and control real-time applications on the target computer with scope objects to collect signal data.

See “Function Reference” and “Functions” for a reference of the target functions.

An understanding of the target object properties and methods will help you to control and test your application on the target computer.

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

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

To create a target object,

- 1 Build a target application. The xPC Target software creates a target object during the build process.

- 2 Use the target object constructor function `xpctarget.xpc`. In the MATLAB Command window, type `tg = xpctarget.xpc`.

Target objects are of class `xpctarget.xpc Class`. A target object has associated properties and methods specific to that object.

## Accessing Help for Target Objects

See “Function Reference” and “Functions” for a reference of the target object functions.

The target application object methods allow you to control a target application on the target computer from the host computer. You enter target application object methods in the MATLAB window on the host computer or use MATLAB code scripts. To access the help for these methods, use the syntax

```
help xpctarget.xpc/method_name
```

If you want to control the target application from the target computer, use target computer commands. See Chapter 10, “Execution Using the Target Computer Command Line”.

## Creating Target Objects

To create a target object, perform the following

- 1 Build a target application. The xPC Target software creates a target object during the build process.
- 2 To create a single target object, or to create multiple target objects in your system, use the target object constructor function `xpctarget.xpc` with arguments. For example, the following creates a target object connected to the host through an RS-232 connection. In the MATLAB Command Window, type:

```
tg = xpctarget.xpc('rs232', 'COM1', '115200')
```

The resulting target object is `tg`.

---

**Tip** Using this method clarifies which target object is associated with a particular target computer.

---

- 3** To check a connection between a host and a target, use the target function `xpctarget.xpc.targetping`. For example, type:

```
tg.targetping
```

- 4** To create a single target object, or to create the first of many targets in your system, use the target object constructor function `xpctarget.xpc` without arguments. For example, in the MATLAB Command Window, type:

```
tg = xpctarget.xpc
```

The resulting target object is `tg`.

---

**Note** If you use `xpctarget.xpc` without arguments to create a target object, use xPC Target Explorer to configure your target computer. This clarifies which target object is associated with a particular target computer.

---

## Displaying Target Object Properties

You might want to list the target object properties to monitor a target application. The properties include the execution time and the 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 computer, and MATLAB displays a list of the target object properties with the updated values.

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

## 2 Type

```
+tg
```

The Status property changes from stopped to running, and the log properties change to Acquiring.

For a list of target object properties with a description, see the target object function `xpctarget.xpc.get` (target application object).

## Setting Target Object Properties from the Host Computer

You can change a target object property by using the xPC Target software `set` method or the dot notation on the host computer. (See “User Interaction” in the *xPC Target Getting Started Guide* guide for limitations on target property changes to sample times.)

With the xPC Target software you can use either a function syntax or an object property syntax to change the target object properties. The syntax `set(target_object, property_name, new_property_value)` can be replaced by

```
target_object.property_name = new_property_value
```

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

- In the MATLAB window, type

```
tg.stoptime = 1000
```

- Alternatively, you can type

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

When you change a target object property, the new property value is downloaded to the target computer. 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`.

### 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 the xPC Target software 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.property_name
```

For example, to access the stop time,

- In the MATLAB window, type

```
endrun = tg.stoptime
```

- Alternatively, you can type

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

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

Signals are not target object properties. To get the value of the Integrator1 signal from the model `xpcosc`,

- In the MATLAB window, type

```
outputvalue = getsignal (tg,0)
```

where 0 is the signal index.

- Alternatively, you can type

```
tg.getsignal(0)
```

---

**Note** Method names are case sensitive. You must type the entire name. Property names are not case sensitive. You do not need to type the entire name as long as the characters you do type are unique for the property.

---

## 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, you must enter method names in full, and in lowercase. For example, to add a target scope with a scope index of 1,

- In the MATLAB window, type

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

- Alternatively, you can type

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

## Target Scope Objects

### In this section...

“What Is a Scope Object?” on page 7-8

“Accessing Help for Scope Objects” on page 7-10

“Displaying Scope Object Properties for a Single Scope” on page 7-10

“Displaying Scope Object Properties for All Scopes” on page 7-11

“Setting the Value of a Scope Property” on page 7-11

“Getting the Value of a Scope Property” on page 7-12

“Using the Method Syntax with Scope Objects” on page 7-13

“Acquiring Signal Data with File Scopes” on page 7-14

“Acquiring Signal Data into Multiple, Dynamically Named Files with File Scopes” on page 7-15

“Advanced Data Acquisition Topics” on page 7-17

### What Is a Scope Object?

The xPC Target software uses scope objects to represent scopes on the target computer. Use scope object functions to view and collect signal data.

See “Function Reference” and “Functions” for a reference of the scope functions.

The xPC Target software uses scopes and scope objects as an alternative to using Simulink scopes and external mode. A scope can exist as part of a Simulink model system or outside a model system.

- A scope that is part of a Simulink model system is a scope block. You add an xPC Target scope block to the model, build an application from that model, and download that application to the target computer.
- A scope that is outside a model is not a scope block. For example, if you create a scope with the `xpctarget.xpc.addscope` method, that scope is not part of a model system. You add this scope to the model after the model has been downloaded and initialized.



This difference affects when and how the scope executes to acquire data.

Scope blocks inherit sample times. A scope block in the root model or a normal subsystem executes at the sample time of its input signals. A scope block in a conditionally executed (triggered/enabled) subsystem executes whenever the containing subsystem executes. Note that in the latter case, the scope might acquire samples at irregular intervals.

A scope that is not part of a model always executes at the base sample time of the model. Thus, it might acquire repeated samples. For example, if the model base sample time is 0.001, and you add to the scope a signal whose sample time is 0.005, the scope will acquire five identical samples for this signal, and then the next five identical samples, and so on.

Understanding the structure of scope objects will help you to use the MATLAB command-line interface to view and collect signal data. A scope object on the host computer represents a scope on the target computer. 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 `xpctarget.xpc.getscope` to create a scope object.
- Use the target object method `xpctarget.xpc.addscope` to create a scope, create a scope object, and assign the scope properties to the scope object.

Upon creation, the xPC Target software assigns the required scope object class for the scope type:

- Target scopes — `xpctarget.xpcsctg` Class, created by calling `xpctarget.xpc.getscope` with scope type `target`
- Host scopes — `xpctarget.xpcscho` Class, created by calling `xpctarget.xpc.getscope` with scope type `host`
- File scopes — `xpctarget.xpcf` Class, created by calling `xpctarget.xpc.getscope` with scope type `file`

A scope object has associated properties and methods specific to that scope type. All of these scope types are based on a common type, `xpctarget.xpcsc Class`, that encompasses the object properties and methods common to all scope object data types. The xPC Target software creates this object if you create multiple scopes of different types for one model and combine those scopes, for example, into a scope vector.

### Accessing Help for Scope Objects

See “Function Reference” and “Functions” for a reference of the scope object functions.

The scope object methods allow you to control scopes on your target computer.

If you want to control the target application from the target computer, use target computer commands. See Chapter 10, “Execution Using the Target Computer Command Line”.

### 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.getscope(1)
```

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

**2** Type

```
sc1
```

The current scope properties are uploaded to the host computer, 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 the target function `xpctarget.xpc.get` (target application object).

## Displaying Scope Object Properties for All Scopes

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

- In the MATLAB window, type

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

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

- Alternatively, type

```
allscopes = getscope(tg)
```

or

```
allscopes = tg.getscope
```

The current scope properties are uploaded to the host computer, 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 first and third scopes, type `allscopes([1,3])`.

For a list of target object properties with a description, see the target function `xpctarget.xpc.get` (target application object).

## Setting the Value of a Scope Property

With the xPC Target software 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(index_vector).property_name = new_property_value
```

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

- In the MATLAB window, type

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

- Alternatively, you can type

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

or

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

Note that you cannot use dot notation to set vector object properties. To assign properties to a vector of scopes, use the `set` method. For example, assume you have a variable `sc12` for two scopes, 1 and 2. To set the `NumSamples` property of these scopes to 300,

- 1 In the MATLAB window, type

```
set(sc12,'NumSamples',300)
```

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

---

**Note** Method names are case sensitive. You must type the entire name. Property names are not case sensitive. You do not need to type the entire name as long as the characters you do type are unique for the property.

---

### 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 the xPC Target software 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(index_vector).property_name
```

For example, to assign the number of samples from the scope object `sc1`,

- In the MATLAB window, type

```
numsamples = sc1.NumSamples
```

- Alternatively, you can type

```
numsamples = get(sc1, 'NumSamples')
```

or

```
sc1.get(NumSamples)
```

Note that you cannot use dot notation to get the values of vector object properties. To get properties of a vector of scopes, use the `get` method. For example, assume you have two scopes, 1 and 2, assigned to the variable `sc12`.

To get the value of `NumSamples` for these scopes, in the MATLAB window, type

```
get(sc12, 'NumSamples')
```

You get a result like the following:

```
ans =  
    [300]  
    [300]
```

To get a list of readable properties, type `scope_object`. The property values are listed in the MATLAB window.

---

**Note** Method names are case sensitive. You must type the entire name. Property names are not case sensitive. You do not need to type the entire name as long as the characters you do type are unique for the property.

---

## 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 either

- `scope_object.method_name(argument_list)`
- `scope_object_vector(index_vector).method_name(argument_list)`

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

- In the MATLAB window, type

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

- Alternatively, you can type

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

### Acquiring Signal Data with File Scopes

You can acquire signal data into a file on the target computer. To do so, you add a file scope to the application. After you build an application and download it to the target computer, you can add a file scope to that application.

---

**Note** Remember to start your scope to acquire signal data.

---

For example, to add a file scope named `sc` to the application, and to add signal 4 to that scope:

- 1 In the MATLAB window, type

```
sc=tg.addscope('file')
```

The xPC Target software creates a file scope for the application.

- 2 Type

```
sc.addsignal(4)
```

- 3 To start the scope, type

```
+sc
```

- 4 To start the target application, type

```
+tg
```

The xPC Target software adds signal 4 to the file scope. When you start the scope and application, the scope saves the signal data for signal 4 to a file, by default named `C:\data.dat`.

See “File Scope” on page 5-51 in Chapter 5, “Signals and Parameters” for a description of file scopes.

If you want to acquire signal data into multiple files, see “Acquiring Signal Data into Multiple, Dynamically Named Files with File Scopes” on page 7-15.

## Acquiring Signal Data into Multiple, Dynamically Named Files with File Scopes

You can acquire signal data into multiple, dynamically named files on the target computer. For example, you can acquire data into multiple files to examine one file while the scope continues to acquire data into other files. To acquire data in multiple files, add a file scope to the application. After you build an application and download it to the target computer, you can add a file scope to that application. You can then configure that scope to log signal data to multiple files.

---

**Note** Remember to start your scope to acquire signal data.

---

For example, configure a file scope named `sc` to the application with the following characteristics:

- Logs signal data into up to nine files whose sizes do not exceed 4096 bytes.
- Creates files whose names contain the string `file_.dat`.
- Contains signal 4.

**1** In the MATLAB window, type

```
tg.StopTime=-1;
```

This parameter directs the target application to run indefinitely.

**2** To add a file scope, type

```
sc=tg.addscope('file');
```

- 3** To enable the file scope to create multiple log files, type

```
sc.DynamicFileName='on';
```

Enable this setting to enable logging to multiple files.

- 4** To enable file scopes to collect data up to the number of samples, then start over again, type

```
sc.AutoRestart='on';
```

Use this setting for the creation of multiple log files.

- 5** To limit each log file size to 4096, type

```
sc.MaxWriteFileSize=4096;
```

You must use this property. Set `MaxWriteFileSize` to a multiple of the `WriteSize` property.

- 6** To enable the file scope to create multiple log files with the same name pattern, type

```
sc.Filename='file_<%>.dat';
```

This sequence directs the software to create up to nine log files, `file_1.dat` to `file_9.dat` on the target computer file system.

- 7** To add signal 4 to the file scope, type

```
sc.addsignal(4);
```

- 8** To start the scope, type

```
+sc
```

- 9** To start the target application, type

```
+tg
```

The software creates a log file named `file_1.dat` and writes data to that file. When the size of `file_1.dat` reaches 4096 bytes (value of



MaxWriteFileSize), the software closes the file and creates `file_2.dat` for writing until its size reaches 4096 bytes. The software repeats this sequence until it fills the last log file, `file_9.dat`. If the target application continues to run and collect data after `file_9.dat`, the software reopens `file_1.dat` and continues to log data, overwriting the existing contents. It cycles through the other log files sequentially. If you do not retrieve the data from existing files before they are overwritten, the data is lost.

If you want to acquire signal data into a single file, see “Acquiring Signal Data with File Scopes” on page 7-14.

## Advanced Data Acquisition Topics

The moment that an xPC Target scope begins to acquire data is user configurable. You can have xPC Target scopes acquire data right away, or define triggers for scopes such that the xPC Target scopes wait until they are triggered to acquire data. You can configure xPC Target scopes to start acquiring data when the following scope trigger conditions are met. These are known as trigger modes.

- **Freerun** — Starts to acquire data as soon as the scope is started (default)
- **Software** — Starts to acquire data in response to a user request. You generate a user request when you call the scope method `xpctarget.xpcsc.trigger` or the scope function `xPCScSoftwareTrigger`.
- **Signal** — Starts to acquire data when a particular signal has crossed a preset level
- **Scope** — Starts to acquire data based on when another (triggering) scope starts

You can use several properties to further refine when a scope acquires data. For example, if you set a scope to trigger on a signal (Signal trigger mode), you can configure the scope to specify the following:

- The signal to trigger the scope (required)
- The trigger level that the signal must cross to trigger the scope to start acquiring data

- Whether the scope should trigger on a rising signal, falling signal, or either one

In the following topics, the trigger point is the sample during which the scope trigger condition is satisfied. For signal triggering, the trigger point is the sample during which the trigger signal passes through the trigger level. At the trigger point, the scope acquires the first sample. By default, scopes start acquiring data from the trigger point onwards. You can modify this behavior using the pre- and posttriggering.

- Pretriggering — Starts to acquire data  $N$  moments before a trigger occurs
- Posttriggering — Starts to acquire data  $N$  moments after a trigger occurs

The `NumPrePostSamples` scope property controls the pre- and posttriggering operation. This property specifies the number of samples to be collected before or after a trigger event.

- If `NumPrePostSamples` is a negative number, the scope is in pretriggering mode, where it starts collecting samples before the trigger event.
- If `NumPrePostSamples` is a positive number, the scope is in a posttriggering mode, where it starts collecting samples after the trigger event.

The following topics describe two examples of acquiring data:

- “Triggering One Scope with Another Scope to Acquire Data” on page 7-18 — Describes a configuration of one scope to trigger another using the concept of pre- and posttriggering
- “Acquiring Gap-Free Data Using Two Scopes” on page 7-21 — Describes how to apply the concept of triggering one scope with another to acquire gap-free data

### **Triggering One Scope with Another Scope to Acquire Data**

This section describes the concept of triggering one scope with another to acquire data. The description uses actual scope objects and properties to describe triggers.

The ability to have one scope trigger another, and to delay retrieving data from the second after a trigger event on the first, is most useful when data

acquisition for the second scope is triggered after data acquisition for the first scope is complete. In the following explanation, Scope 2 is triggered by Scope 1.

- Assume two scopes objects are configured as a vector with the command

```
sc = tg.addscope('host', [1 2]);
```

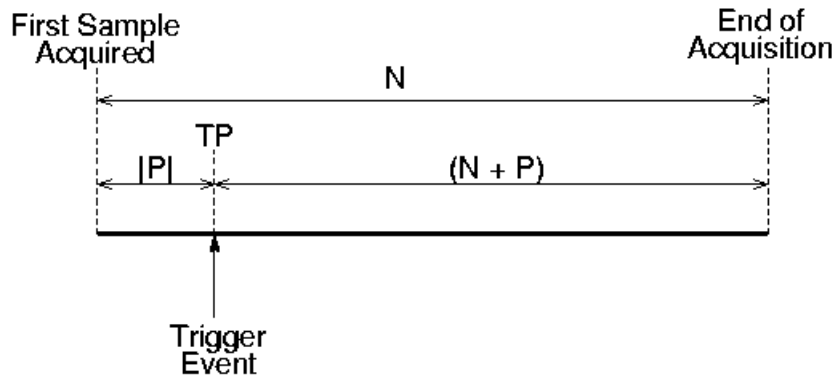
- For Scope 1, set the following values:
  - `sc(1).ScopeId = 1`
  - `sc(1).NumSamples = N`
  - `sc(1).NumPrePostSamples = P`
- For Scope 2, set the following values:
  - `sc(2).ScopeId = 2`
  - `sc(2).TriggerMode = 'Scope'`
  - `sc(2).TriggerScope = 1`
  - `sc(2).TriggerSample = range 0 to (N + P - 1)`

In the figures below, TP is the trigger point or sample where a trigger event occurs. Scope 1 begins acquiring data as described.

In the simplest case, where  $P = 0$ , Scope 1 acquires data right away.

Pretriggering ( $P < 0$ ) on page 7-20 illustrates the behavior if  $P$ , the value of `NumPrePostSamples`, is negative. In this case, Scope 1 starts acquiring data  $|P|$  samples before  $TP$ . Scope 2 begins to acquire data only after  $TP$  occurs.

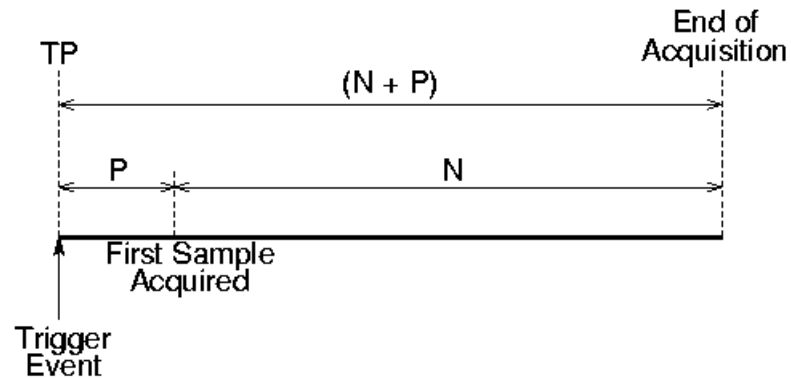
### Pretriggering ( $P < 0$ )



### Pretriggering ( $P < 0$ )

Posttriggering ( $P > 0$ ) on page 7-20 illustrates the behavior if  $P$ , the value of `NumPrePostSamples`, is positive. In this case, Scope 1 starts acquiring data  $|P|$  samples after  $TP$  occurs.

### Posttriggering ( $P > 0$ )



### Posttriggering ( $P > 0$ )

Scope 1 triggers Scope 2 after the trigger event occurs. The following describes some of the ways you can trigger Scope 2:

- `sc(2).TriggerSample = 0` — Causes Scope 2 to be triggered when Scope 1 is triggered. TP for both scopes as at the same sample.
- `sc(2).TriggerSample = n > 0` — Causes TP for Scope 2 to be  $n$  samples after TP for Scope 1. Note that setting `sc(2).TriggerSample` to a value larger than  $(N + P - 1)$  does not cause an error; it implies that Scope 2 will never trigger, since Scope 1 will never acquire more than  $(N + P - 1)$  samples after TP.
- `sc(2).TriggerSample = 0 < n < (N + P)` — Enables you to obtain some of the functionality that is available with pre- or posttriggering. For example, if you have the following Scope 1 and Scope 2 settings,
  - Scope 1 has `sc(1).NumPrePostSamples = 0` (no pre- or posttriggering)
  - Scope 2 has `sc(2).TriggerSample = 10`
  - Scope 2 has `sc(2).NumPrePostSample = 0`

The behavior displayed by Scope 2 is equivalent to having `sc(2).TriggerSample = 0` and `sc(2).NumPrePostSamples = 10`.

- `sc(2).TriggerSample = -1` — Causes Scope 2 to start acquiring data from the sample after Scope 1 stops acquiring.

---

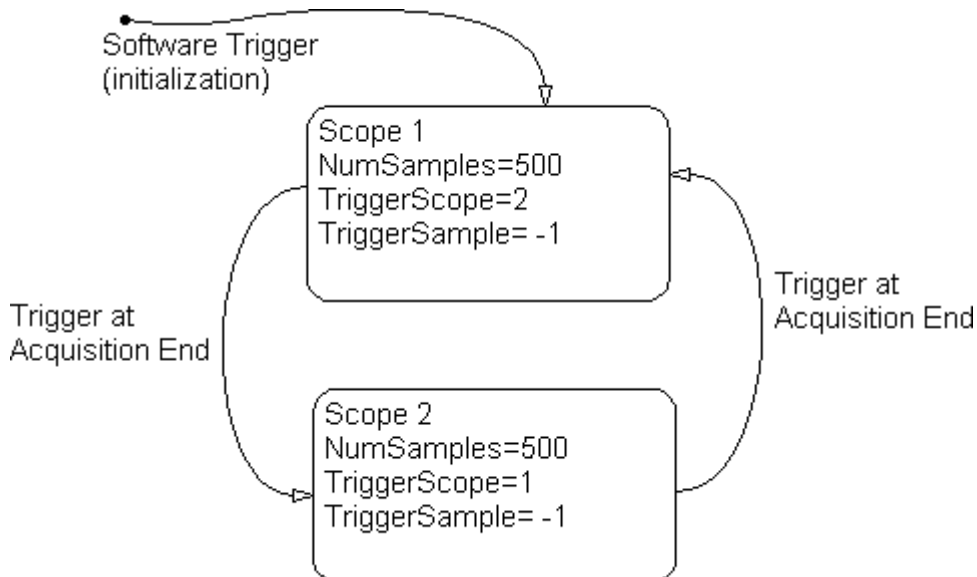
**Note** The difference between setting `TriggerSample = (N + P - 1)`, where  $N$  and  $P$  are the parameters of the triggering scope (Scope 1) and `TriggerSample = -1` is that in the former case, the first sample of Scope 2 will be at the same time as the last sample of Scope 1, whereas in the latter, the first sample of Scope 2 will be one sample after the last sample of Scope 1. This means that in the former case both scopes acquire simultaneously for one sample, and in the latter they will never simultaneously acquire.

---

### Acquiring Gap-Free Data Using Two Scopes

With two scopes, you can acquire gap-free data. Gap-free data is data that two scopes acquire consecutively, with no overlap. The first scope acquires data up to  $N$ , then stops. The second scope begins to acquire data at  $N+1$ . This is functionality that you cannot achieve through pre- or posttriggering.

Acquisition of Gap-Free Data on page 7-22 graphically illustrates how scopes trigger one another. In this example, the `TriggerMode` property of Scope 1 is set to 'Software'. This allows Scope 1 to be software triggered to acquire data when it receives the command `sc1.trigger`.



### Acquisition of Gap-Free Data

The following procedure describes how you can programmatically acquire gap-free data with two scopes.

- 1 Build and download the Simulink model `xpcosc.mdl` to the target computer.
- 2 In the MATLAB Command Window, assign `tg` to the target computer and set the `StopTime` property to 1. For example,

```
tg=xpctarget.xpc
tg.StopTime = 1;
```

- 3 Add two host to the target application. You can assign the two scopes to a vector, `sc`, so that you can work with both scopes with one command.

```
sc = tg.addscope('host', [1 2]);
```

- 4** Add the signals of interest (0 and 1) to both scopes.

```
addsignal(sc,[0 1]);
```

- 5** Set the NumSamples property for both scopes to 500 and the TriggerSample property for both scopes to -1. With this property setting, each scope triggers the next scope *at the end* of its 500 sample acquisition.

```
set(sc, 'NumSamples', 500, 'TriggerSample', -1)
```

- 6** Set the TriggerMode property for both scopes to 'Scope'. Set the TriggerScope property such that each scope is triggered by the other.

```
set(sc, 'TriggerMode', 'Scope');
sc(1).TriggerScope = 2;
sc(2).TriggerScope = 1;
```

- 7** Set up storage for time, t, and signal, data acquisition.

```
t = [];
data = zeros(0, 2);
```

- 8** Start both scopes and the model.

```
start(sc);
start(tg);
```

Note that both scopes receive exactly the same signals, 0 and 1.

- 9** Trigger scope 1 to start acquiring data.

```
scNum = 1;
sc(scNum).trigger;
```

Setting scNum to 1 indicates that scope 1 will acquire data first.

- 10** Start acquiring data using the two scopes to double buffer the data.

```
while (1)
    % Wait until this scope has finished acquiring 500 samples
    % or the model stops (scope is interrupted).
    while ~(strcmp(sc(scNum).Status, 'Finished') || ...
            strcmp(sc(scNum).Status, 'Interrupted')), end
```

```
% Stop buffering data when the model stops.
if strcmp(tg.Status, 'stopped')
    break
end
% Save the data.
t(end + 1 : end + 500) = sc(scNum).Time;
data(end + 1 : end + 500, :) = sc(scNum).Data;
% Restart this scope.
start(sc(scNum));
% Switch to the next scope.
%Shortcut for if(scNum==1) scNum=2;else scNum=1,end
scNum = 3 - scNum;
end
```

**11** When done, remove the scopes.

```
% Remove the scopes we added.
remscope(tg,[1 2]);
```

The following is a complete code listing for the preceding double-buffering data acquisition procedure. You can copy and paste this code into a MATLAB file and run it after you download the model (`xpcosc.mdl`) to the target computer. This example assumes that the communication speed between the host and target computer is fast enough to handle the number of samples and can acquire the full data set before the next acquisition cycles starts. In a similar way, you can use more than two scopes to implement a triple- or quadruple-buffering scheme.

```
% Assumes model xpcosc.mdl has been built and loaded on the target computer.
% Attach to the target computer and set StopTime to 1 sec.
tg = xpctarget.xpc;
tg.StopTime = 1;
% Add two host scopes.
sc = tg.addscope('host', [1 2]);
% [0 1] are the signals of interest. Add to both scopes.
addsignal(sc,[0 1]);
% Each scope triggers next scope at end of a 500 sample acquisition.
set(sc, 'NumSamples', 500, 'TriggerSample', -1);
set(sc, 'TriggerMode', 'Scope');
sc(1).TriggerScope = 2;
sc(2).TriggerScope = 1;
```



```
% Initialize time and data log.
t = [];
data = zeros(0, 2);
% Start the scopes and the model.
start(sc);
start(tg);
% Start things off by triggering scope 1.
scNum = 1;
sc(scNum).trigger;
% Use the two scopes as a double buffer to log the data.
while (1)
    % Wait until this scope has finished acquiring 500 samples
    % or the model stops (scope is interrupted).
    while ~(strcmp(sc(scNum).Status, 'Finished') || ...
            strcmp(sc(scNum).Status, 'Interrupted')), end
        % Stop buffering data when the model stops.
        if strcmp(tg.Status, 'stopped')
            break
        end
        % Save the data.
        t(end + 1 : end + 500) = sc(scNum).Time;
        data(end + 1 : end + 500, :) = sc(scNum).Data;
        % Restart this scope.
        start(sc(scNum));
        % Switch to the next scope.
        scNum = 3 - scNum;
    end
% Remove the scopes we added.
remscope(tg,[1 2]);
% Plot the data.
plot(t,data); grid on; legend('Signal 0','Signal 1');
```



# Logging Signal Data with FTP and File System Objects

---

- “File Systems” on page 8-2
- “FTP and File System Objects” on page 8-4
- “Using xpctarget.ftp Objects” on page 8-5
- “Using xpctarget.fs Objects” on page 8-10

## File Systems

xPC Target file scopes create files on the target computer. To work with these files from the host computer, you need to work with the `xpctarget.ftp` and `xpctarget.fs` objects. The `xpctarget.ftp` object allows you to perform basic file transfer operations on the target computer file system. The `xpctarget.fs` object allows you to perform file system-like operations on the target computer file system.

You cannot direct the scope to write the data to a file on the xPC Target host computer. Once the software has written the signal data file to the target computer, you can access the contents of the file for plotting or other inspection from the host computer. The software can write data files to

- The C:\ or D:\ drive of the target computer. This can be a serial ATA (SATA) or parallel ATA (PATA)/Integrated Device Electronics (IDE) drive. The xPC Target software supports file systems of type FAT-12, FAT-16, or FAT-32. Verify that the hard drive is not cable-selected and that the BIOS can detect it. The type of file system (FAT-12, FAT-16, or FAT-32) limits the maximum size of the file. The target computer file system uses the 8.3 file name convention. This means that a target computer file name cannot exceed eight characters. Its file extension cannot exceed 3 characters.

If you have a target computer with multiple partitions on a hard drive, the xPC Target software file scope can access those partitions if they are formatted with FAT-12, FAT-16, or FAT-32. It will ignore any unsupported file systems.

- A 3.5-inch disk drive.
- Disks connected to a secondary IDE controller. The software supports up to four drives through the second IDE controller. By default, it works with drives configured as the primary master. If you want to use a secondary IDE controller, you must configure the xPC Target software for it (see “Converting xPC Target File Format Content to Double Precision Data” on page 8-14 in Chapter 4, “Target Application Environment”). The software searches for another drive in the first four ports of the target computer.

The largest single file that you can create is 4 GB.

Note that writing data files to 3.5-inch disk drives is considerably slower than writing to hard drives.

You can access signal data files, or any target computer system file, in one of the following ways:

- If you are running the target computer as a standalone system, you can access that file by rebooting the target computer under an operating system such as DOS and accessing the file through the operating system utilities.
- If you are running the target computer in conjunction with a host computer, you can access the target computer file from the host computer by representing that file as an `xpctarget.ftp` object. Through the MATLAB interface, use `xpctarget.ftp` methods on that FTP object. The `xpctarget.ftp` object methods are file transfer operations such as `get` and `put`.
- If you are running the target computer in conjunction with a host computer, you can access the target computer file from the host computer by representing the target computer file system as an `xpctarget.fs` object. Through the MATLAB interface, use the `xpctarget.fs` methods on the file system and perform file system-like methods such as `fopen` and `fread` on the signal data file. These methods work like the MATLAB file I/O methods. The `xpctarget.fs` methods also include file system utilities that allow you to collect target computer file system information for the disk and disk buffers.

This topic describes procedures on how to use the `xpctarget.ftp` and `xpctarget.fs` methods for common operations. See “Function Reference” and “Functions” for a reference of the methods for these objects.

---

**Note** This topic focuses primarily on working with the target computer data files that you generate from an xPC Target scope object of type `file`.

---

For a demo of how to perform data logging with file scopes, see [Data Logging With a File Scope](#).

## FTP and File System Objects

The xPC Target software uses two objects, `xpctarget.ftp` and `xpctarget.fs` (file system), to work with files on a target computer. You use the `xpctarget.ftp` object to perform file transfer operations between the host and target computer. You use the `xpctarget.fs` object to access the target computer file system. For example, you can use an `xpctarget.fs` object to open, read, and close a signal data file created by an xPC Target file scope.

---

**Note** This feature provides FTP-like commands, such as `get` and `put`. However, it is not a standard FTP implementation. For example, the software does not support the use of a standard FTP client.

---

To create an `xpctarget.ftp` object, use the FTP object constructor function `xpctarget.ftp`. In the MATLAB Command Window, type

```
f = xpctarget.ftp
```

The xPC Target software uses a file system object on the host computer to represent the target computer file system. You use file system objects to work with that file system from the host computer.

To create an `xpctarget.fs` object, use the FTP object constructor function `xpctarget.fs`. In the MATLAB window, type

```
f = xpctarget.fs
```

Both `xpctarget.ftp` and `xpctarget.fs` belong to the `xpctarget.fsbase` object. This object encompasses the methods common to `xpctarget.ftp` and `xpctarget.fs`. You can call the `xpctarget.fsbase` methods for both `xpctarget.ftp` and `xpctarget.fs` objects. The xPC Target software creates the `xpctarget.fsbase` object when you create either an `xpctarget.ftp` or `xpctarget.fs` object. You enter `xpctarget.fsbase` object methods in the MATLAB Command Window on the host computer or use MATLAB code scripts.

## Using xpctarget.ftp Objects

### In this section...

“Overview” on page 8-5

“Accessing Files on a Specific Target Computer” on page 8-6

“Listing the Contents of the Target Computer Folder” on page 8-7

“Retrieving a File from the Target Computer to the Host Computer” on page 8-8

“Copying a File from the Host Computer to the Target Computer” on page 8-8

### Overview

The `xpctarget.ftp` object enables you to work with any file on the target computer, including the data file that you generate from an xPC Target scope object of type `file`. You enter target object methods in the MATLAB window on the host computer or use scripts. The `xpctarget.ftp` object has methods that allow you to use

- `xpctarget.fsbase.cd` to change directories
- `xpctarget.fsbase.dir` to list the contents of the current folder
- `xpctarget.fsbase.mkdir` to make a folder
- `xpctarget.fsbase.pwd` to get the current working folder path
- `xpctarget.fsbase.rmdir` to remove a folder
- `xpctarget.ftp.get (ftp)` to retrieve a file from the target computer to the host computer
- `xpctarget.ftp.put` to place a file from the host computer to the target computer

The procedures in this section assume that the target computer has a signal data file created by an xPC Target file scope. This file has the pathname `C:\data.dat`. See “Creating a Simple Simulink Model” and “Signal Tracing with xPC Target Scope Blocks” on page 5-49 for additional details.

The xPC Target software also provides methods that allow you to perform file system-type operations, such as opening and reading files. For a complete list of these methods, see “Using xpctarget.fs Objects” on page 8-10.

## Accessing Files on a Specific Target Computer

You can access specific target computer files from the host computer for the `xpctarget.ftp` object.

Use the `xpctarget.ftp` creator function. If your system has multiple targets, you can access specific target computer files from the host computer for the `xpctarget.ftp` object.

For example, to list the name of the current folder of a target computer through a TCP/IP connection,

- 1 In the MATLAB Command Window, type a command like the following to assign the `xpctarget.ftp` object to a variable.

```
f=xpctarget.ftp('TCPIP','192.168.0.10','22222');
```

- 2 Type

```
f.pwd;
```

Alternatively, you can use the `xpctarget.xpc` constructor to first construct a target object, then use that target object as an argument to `xpctarget.ftp`.

- 1 In the MATLAB window, type a command like the following to assign the `xpctarget.xpc` object to a variable.

```
tg1=xpctarget.xpc('TCPIP','192.168.0.10','22222');
```

- 2 Type the following command to assign the `xpctarget.ftp` object to the `tg1` target object variable.

```
f=xpctarget.ftp(tg1);
```

Alternatively, if you want to work with the files of the default target computer, you can use the `xpctarget.ftp` constructor without arguments.



In the MATLAB window, type a command like the following to assign the `xpctarget.ftp` object to a variable.

```
f=xpctarget.ftp;
```

The xPC Target software assigns the `f` variable to the default target computer.

## Listing the Contents of the Target Computer Folder

You can list the contents of the target computer folder by using xPC Target methods on the host computer for the `xpctarget.ftp` object. Use the method syntax to run an `xpctarget.ftp` object method:

```
method_name(ftp_object)
```

---

**Note** You must use the `dir(f)` syntax to list the contents of the folder. To get the results in an M-by-1 structure, use a syntax like `y=dir(f)`. See the `xpctarget.fsbase.dir` method reference for further details.

---

For example, to list the contents of the `C:\` drive,

- 1 In the MATLAB window, type the following to assign the `xpctarget.ftp` object to a variable:

```
f=xpctarget.ftp;
```

- 2 Type

```
f.pwd
```

This gets the current folder. You get a result like the following:

```
ans =  
C:\
```

- 3 Type the following to list the contents of this folder:

```
dir(f)
```

## Retrieving a File from the Target Computer to the Host Computer

You can retrieve a copy of a data file from the target computer by using xPC Target methods on the host computer for the `xpctarget.ftp` object.

Use the method syntax to run an `xpctarget.ftp` object method. The syntax `method_name(ftp_object, argument_list)` can be replaced with

```
ftp_object.method_name(argument_list)
```

For example, to retrieve a file named `data.dat` from the target computer C:\ drive (default),

- 1 If you have not already done so, in the MATLAB window, type the following to assign the `xpctarget.ftp` object to a variable.

```
f=xpctarget.ftp;
```

- 2 Type

```
f.get('data.dat');
```

This retrieves the file and saves that file to the variable `data`. This content is in the xPC Target file format.

## Copying a File from the Host Computer to the Target Computer

You can place a copy of a file from the host computer by using xPC Target methods on the host computer for the `xpctarget.ftp` object.

Use the method syntax to run an `xpctarget.ftp` object method. The syntax `method_name(ftp_object, argument_list)` can be replaced with

```
ftp_object.method_name(argument_list)
```

For example, to copy a file named `data2.dat` from the host computer to the target computer C:\ drive (default),

- 1 If you have not already done so, in the MATLAB window, type the following to assign the `xpctarget.ftp` object to a variable.

```
f=xpctarget.ftp;
```

**2** Type the following to save that file to the variable data.

```
f.put('data2.dat');
```

## Using `xpctarget.fs` Objects

### In this section...

“Overview” on page 8-10

“Accessing File Systems from a Specific Target Computer” on page 8-11

“Retrieving the Contents of a File from the Target Computer to the Host Computer” on page 8-12

“Removing a File from the Target Computer” on page 8-15

“Getting a List of Open Files on the Target Computer” on page 8-16

“Getting Information about a File on the Target Computer” on page 8-17

“Getting Information about a Disk on the Target Computer” on page 8-18

### Overview

The `fs` object enables you to work with the target computer file system from the host computer. You enter target object methods in the MATLAB window on the host computer or use scripts. The `fs` object has methods that allow you to use

- `xpctarget.fsbase.cd` to change directories
- `xpctarget.fsbase.dir` to list the contents of the current folder
- `xpctarget.fsbase.mkdir` to make a folder
- `xpctarget.fsbase.pwd` to get the current working folder path
- `xpctarget.fsbase.rmdir` to remove a folder
- `xpctarget.fs.diskinfo` to get information about the specified disk
- `xpctarget.fs.fclose` to close a file (similar to MATLAB `fclose`)
- `xpctarget.fs.fileinfo` to get information about a particular file
- `xpctarget.fs.filetable` to get information about files in the file system
- `xpctarget.fs.fopen` to open a file (similar to MATLAB `fopen`)
- `xpctarget.fs.fread` to read a file (similar to MATLAB `fread`)

- `xpctarget.fs.fwrite` to write a file (similar to MATLAB `fwrite`)
- `xpctarget.fs.getfilesize` to get the size of a file in bytes
- `xpctarget.fs.removefile` to remove a file from the target computer

Useful global utility:

- `readxpcfile`, to interpret the raw data from the `fread` method

The procedures in this section assume that the target computer has a signal data file created by an xPC Target file scope. This file has the pathname `C:\data.dat`.

The xPC Target software also provides methods that allow you to perform file transfer operations, such as putting files on and getting files from a target computer. For a description of these methods, see “Using `xpctarget.ftp` Objects” on page 8-5.

## Accessing File Systems from a Specific Target Computer

You can access specific target computer files from the host computer for the `xpctarget.fs` object.

Use the `xpctarget.fs` creator function. If your system has multiple targets, you can access specific target computer files from the host computer for the `xpctarget.fs` object.

For example, to list the name of the current folder of a target computer through a TCP/IP connection,

- 1 In the MATLAB window, type a command like the following to assign the `xpctarget.fs` object to a variable.

```
fsys=xpctarget.fs('TCP/IP','192.168.0.10','22222');
```

- 2 Type

```
fsys.dir;
```

Alternatively, you can use the `xpctarget.xpc` constructor to first construct a target object, then use that target object as an argument to `xpctarget.fs`.

- 1 In the MATLAB window, type a command like the following to assign the `xpctarget.xpc` object to a variable.

```
tg1=xpctarget.xpc('TCPIP','192.168.0.10','22222');
```

- 2 Type the following command to assign the `xpctarget.fs` object to the `tg1` target object variable.

```
fs=xpctarget.fs(tg1);
```

Alternatively, if you want to work with the file system of the default target computer, you can use the `xpctarget.fs` constructor without arguments.

- 1 In the MATLAB window, type a command like the following to assign the `xpctarget.fs` object to a variable.

```
fsys=xpctarget.fs;
```

The xPC Target software assigns the `fsys` variable to the default target computer.

- 2 Type

```
fsys.dir;
```

## Retrieving the Contents of a File from the Target Computer to the Host Computer

You can retrieve the contents of a data file from the target computer by using xPC Target methods on the host computer for the `xpctarget.fs` object. This is an alternate method to “Exporting Data from File Scopes to MATLAB Workspace” on page 5-36 in Chapter 5, “Signals and Parameters”.

Use the method syntax to run an `xpctarget.fs` object method. The syntax `method_name(fs_object, argument_list)` can be replaced with

```
fs_object.method_name(argument_list)
```

For example, to retrieve the contents of a file named `data.dat` from the target computer `C:\ drive` (default),

- 1 If you have not already done so, in the MATLAB window, type the following to assign the `xpctarget.fs` object to a variable.

```
fsys=xpctarget.fs;
```

- 2 Type

```
h=fsys.fopen('data.dat');
```

or

```
h=fopen(fsys,'data.dat');
```

This opens the file `data.dat` for reading and assigns the file identifier to `h`.

- 3 Type

```
data2=fsys.fread(h);
```

or

```
data2=fread(fsys,h);
```

This reads the file `data.dat` and stores the contents of the file to `data2`. This content is in the xPC Target file format.

- 4 Type

```
fsys fclose(h);
```

This closes the file `data.dat`.

Before you can view or plot the contents of this file, you must convert the contents. See “Converting xPC Target File Format Content to Double Precision Data” on page 8-14.

## Converting xPC Target File Format Content to Double Precision Data

The xPC Target software provides the script `readxpcfile.m` to convert xPC Target file format content (in bytes) to double precision data representing the signals and timestamps. The `readxpcfile.m` script takes in data from a file in xPC Target format. The data must be a vector of bytes (`uint8`). To convert the data to `uint8`, use a command like the following:

```
data2 = uint8(data2');
```

This section assumes that you have a variable, `data2`, that contains data in the xPC Target file format (see “Retrieving the Contents of a File from the Target Computer to the Host Computer” on page 8-12):

- 1 In the MATLAB window, change folder to the folder that contains the xPC Target format file.
- 2 Type

```
new_data2=readxpcfile(data2);
```

The `readxpcfile` script converts the format of `data2` from the xPC Target file format to an array of bytes. It also creates a structure for that file in `new_data2`, of which one of the elements is an array of doubles, `data`. The `data` member is also appended with a time stamp vector. All data is returned as doubles, which represent the real-world values of the original Simulink signals at the specified times during target execution.

You can view or examine the signal data. You can also plot the data with `plot(new_data2.data)`.

If you are using the xPC Target software in StandAlone mode, you can extract the data from the data file if you know the number of signals in the scope and file header size. If you know these numbers, you can extract the data. Note the following:

- First determine the file header size. To obtain the file header size, ignore the first eight bytes of the file. The next four bytes store the header size as an unsigned integer.



- After the header size number of bytes, the file stores the signals sequentially as doubles. For example, assume the scope has three signals,  $x$ ,  $y$ , and  $z$ . Assume that  $x[0]$  is the value of  $x$  at sample 0,  $x[1]$  is the value at sample 1, and so forth, and  $t[0]$ ,  $t[1]$  are the simulation time values at samples 0, 1, and so forth, respectively. The file saves the data using the following pattern:

```
x[0] y[0] z[0] t[0] x[1] y[1] z[1] t[1] x[2] y[2] z[2] t[2]...
x[N] y[N] z[N] t[N]
```

$N$  is the number of samples acquired. The file saves  $x$ ,  $y$ ,  $z$ , and  $t$  as doubles at 8 bytes each.

## Removing a File from the Target Computer

You can remove a file from the target computer by using xPC Target methods on the host computer for the `xpctarget.fs` object. If you have not already done so, close this file first with `fclose`.

Use the method syntax to run an `xpctarget.fs` object method. The syntax `method_name(fs_object, argument_list)` can be replaced with

```
fs_object.method_name(argument_list)
```

For example, to remove a file named `data2.dat` from the target computer C:\ drive (default),

- 1 If you have not already done so, in the MATLAB window, type the following to assign the `xpctarget.fs` object to a variable.

```
fsys=xpctarget.fs;
```

- 2 Type the following to remove the specified file from the target computer.

```
fsys.removefile('data2.dat');
```

or

```
removefile(fsys,'data2.dat');
```

## Getting a List of Open Files on the Target Computer

You can get a list of open files on the target computer file system from the host computer by using xPC Target methods on the host computer for the `xpctarget.fs` object. Do this to identify files you can close. The target computer file system limits the number of open files you can have to eight.

Use the method syntax to run an `xpctarget.fs` object method. The syntax `method_name(fs_object, argument_list)` can be replaced with

```
fs_object.method_name(argument_list)
```

For example, to get a list of open files for the file system object `fsys`,

- 1 If you have not already done so, in the MATLAB window, type the following to assign the `xpctarget.fs` object to a variable.

```
fsys=xpctarget.fs;
```

- 2 Type

```
fsys.filetable
```

If the file system has open files, a list like the following is displayed:

```
ans =
  Index      Handle  Flags      FilePos  Name
-----
      0  00060000  R__         8512  C:\DATA.DAT
      1  00080001  R__           0  C:\DATA1.DAT
      2  000A0002  R__         8512  C:\DATA2.DAT
      3  000C0003  R__         8512  C:\DATA3.DAT
      4  001E0001  R__           0  C:\DATA4.DA
```

- 3 The table returns the open file handles in hexadecimal. To convert a handle to one that other `xpctarget.fs` methods, such as `fclose`, can use, use the `hex2dec` function. For example,

```
h1 = hex2dec('001E0001')
h1 =
1966081
```

**4** To close that file, use the `xpctarget.fs fclose` method. For example,

```
fsys fclose(h1);
```

## Getting Information about a File on the Target Computer

You can display information for a file on the target computer file system from the host computer by using xPC Target methods on the host computer for the `xpctarget.fs` object.

Use the method syntax to run an `xpctarget.fs` object method. The syntax `method_name(fs_object, argument_list)` can be replaced with

```
fs_object.method_name(argument_list)
```

For example, to display the information for the file identifier `fid1`,

**1** If you have not already done so, in the MATLAB window, type the following to assign the `xpctarget.fs` object to a variable.

```
fsys=xpctarget.fs;
```

**2** Type

```
fid1=fsys.fopen('data.dat');
```

This opens the file `data.dat` for reading and assigns the file identifier to `fid1`.

**3** Type

```
fsys.fileinfo(fid1);
```

This returns disk information like the following for the C:\ drive file system.

```
ans =  
      FilePos: 0  
    AllocatedSize: 12288  
    ClusterChains: 1  
VolumeSerialNumber: 1.0450e+009  
      FullName: 'C:\DATA.DAT'
```

## Getting Information about a Disk on the Target Computer

You can display information for a disk on the target computer file system from the host computer by using xPC Target methods on the host computer for the `xpctarget.fs` object.

Use the method syntax to run an `xpctarget.fs` object method. The syntax `method_name(fs_object, argument_list)` can be replaced with

```
fs_object.method_name(argument_list)
```

For example, to display the disk information for the C:\ drive,

- 1 If you have not already done so, in the MATLAB window, type the following to assign the `xpctarget.fs` object to a variable.

```
fsys=xpctarget.fs;
```

- 2 Type

```
fsys.diskinfo('C:\');
```

This returns disk information like the following for the C:\ drive file system.

```
ans =  
          Label: 'SYSTEM '  
    DriveLetter: 'C'  
      Reserved: ''  
  SerialNumber: 1.0294e+009  
FirstPhysicalSector: 63  
          FATType: 32  
          FATCount: 2  
    MaxDirEntries: 0  
  BytesPerSector: 512  
SectorsPerCluster: 4  
    TotalClusters: 2040293  
      BadClusters: 0  
    FreeClusters: 1007937  
          Files: 19968  
    FileChains: 22480
```

FreeChains: 1300  
LargestFreeChain: 64349



# Execution Using Graphical User Interface Models

---

You can use the Simulink interface to create a custom graphical user interface (GUI) for your xPC Target application. To do this, create an user interface model with the Simulink interface and add-on products like Simulink 3D Animation™ or Altia® Design (a third-party product).

## xPC Target Interface Blocks to Simulink Models

### In this section...

“Simulink User Interface Model” on page 9-2

“Creating a Custom Graphical Interface” on page 9-3

“To xPC Target Block” on page 9-4

“From xPC Target Block” on page 9-5

“Creating a Target Application Model” on page 9-7

“Marking Block Parameters” on page 9-7

“Marking Block Signals” on page 9-10

### Simulink User Interface Model

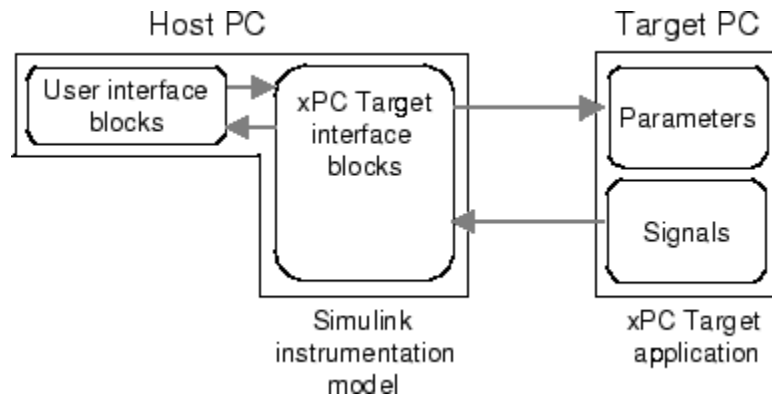
A user interface model is a Simulink model containing Simulink blocks from add-on products and interface blocks from the xPC Target block library. This user interface model can connect to a custom graphical interface using Simulink 3D Animation or Altia products. The user interface model runs on the host computer and communicates with your target application running on the target computer using To xPC Target and From xPC Target blocks.

The user interface allows you to change parameters by downloading them to the target computer, and to visualize signals by uploading data to the host computer.

**Simulink 3D Animation** — The Simulink 3D Animation product enables you to display a Simulink user interface model in 3-D. It provides Simulink blocks that communicate with xPC Target interface blocks. These blocks then communicate to a graphical interface. This graphical interface is a Virtual Reality Modeling Language (VRML) world displayed with a Web browser using a VRML plug-in.

**Altia Design** — Altia also provides Simulink blocks that communicate with xPC Target interface blocks. These blocks then communicate with Altia’s graphical interface or with a Web browser using the Altia ProtoPlay plug-in.





## Creating a Custom Graphical Interface

The xPC Target block library provides Simulink interface blocks to connect graphical interface elements to your target application. The steps for creating your own custom user interface are listed below:

- 1** In the Simulink target application model, decide which block parameters and block signals you want to have access to through graphical interface control devices and graphical interface display devices.
- 2** Tag all block parameters in the Simulink model that you want to be connected to a control device. See “Marking Block Parameters” on page 9-7.
- 3** Tag all signals in Simulink model that you want to be connected to a display device. See “Marking Block Signals” on page 9-10.
- 4** In the MATLAB interface, run the function `xpcsiface('model_name')` to create the user interface template model. This function generates a new Simulink model containing only the xPC Target interface blocks (To xPC Target and From xPC Target) defined by the tagged block parameters and block signals in the target application model.
- 5** To the user interface template model, add Simulink interface blocks from add-on products (Simulink 3D Animation, Altia Design).
  - You can connect Altia blocks to the xPC Target To PC Target interface blocks. To xPC Target blocks on the left should be connected to control devices.

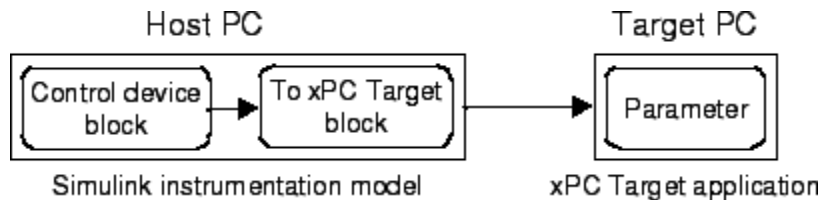
- You can connect Altia and Simulink 3D Animation blocks to the xPC Target From PC Target interface blocks. From xPC Target blocks on the right should be connected to the display devices.

You can position these blocks to your liking.

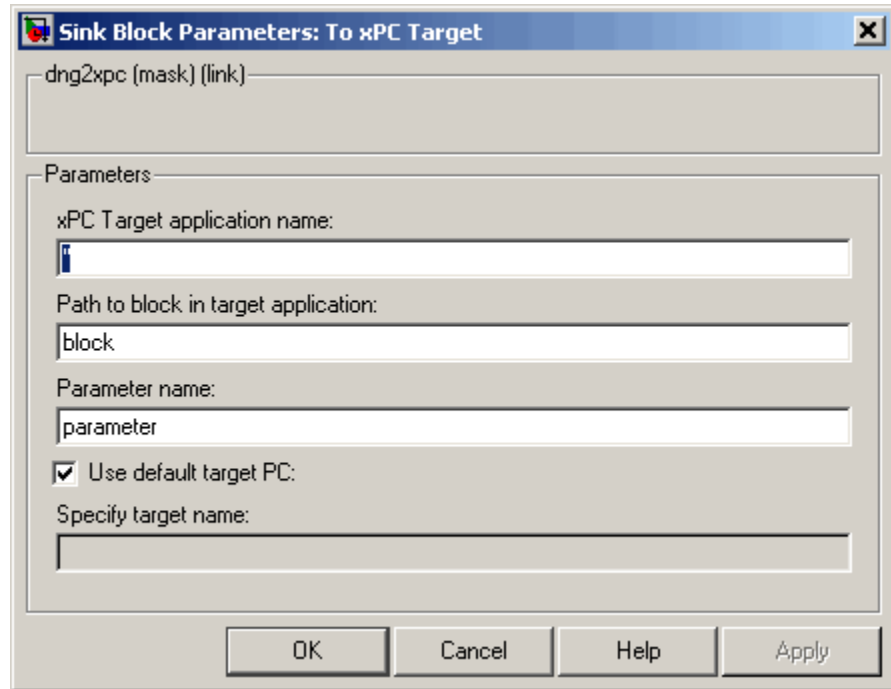
- 6 Start both the xPC Target application and the Simulink user interface model that represents the xPC Target application.

### To xPC Target Block

This block behaves as a sink and usually receives its input data from a control device. The purpose of this block is to write a new value to a specific parameter on the target application.



This block is implemented as a MATLAB S-function. The block is optimized so that it only changes a parameter on the target application when the input value differs from the value that existed at the last time step. This block uses the parameter downloading feature of the xPC Target command-line interface. This block is available from the xpclib/Misc block sublibrary. See To xPC Target in the xPC Target I/O Reference for further configuration details.



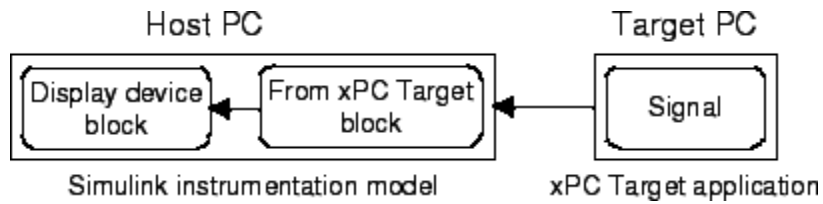
---

**Note** The use of To xPC Target blocks requires a connection between the host and target computer. Operations such as opening a model that contains these blocks or copying these blocks within or between models will take significantly longer than normal without a connection between the host and target computers.

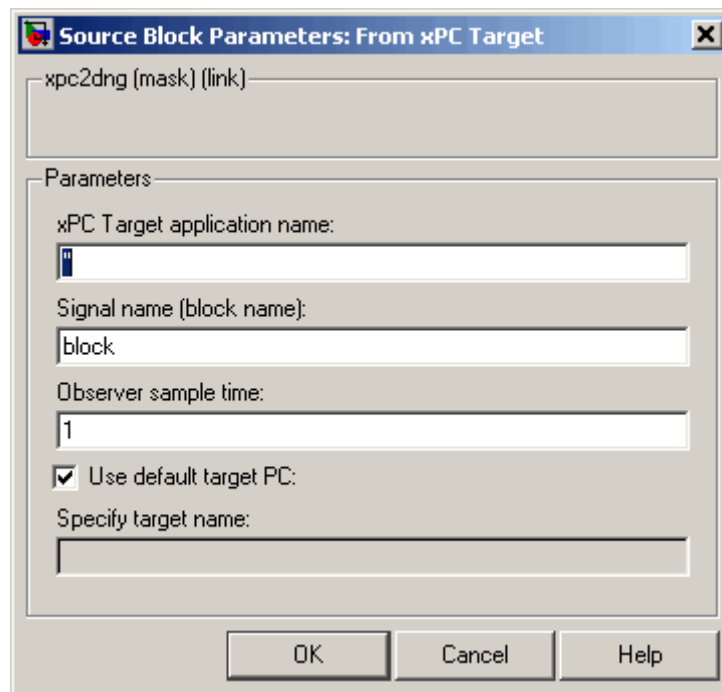
---

## From xPC Target Block

This block behaves like a source and its output is usually connected to the input of a display device.



Because only one numerical value per signal is uploaded during a time step, the number of samples of a scope object is set to 1. The block uses the capability of the xPC Target command-line interface and is implemented as a MATLAB S-function. This block is available from the xpclib/Misc sublibrary. See From xPC Target in the xPC Target I/O Reference for further configuration details.



---

**Note** The use of From xPC Target blocks requires a connection between the host and target computers. Operations such as opening a model that contains these blocks or copying these blocks within or between models will take significantly longer than normal without a connection between the host and target computers.

---

## Creating a Target Application Model

A target application model is a Simulink model that describes your physical system, a controller, and its behavior. You use this model to create a real-time target application, and you use this model to select the parameters and signals you want to connect to a custom graphical interface.

Creating a target application model is the first step you need to do before you can tag block parameters and block signals for creating a custom graphical interface.

See “Marking Block Parameters” on page 9-7 and “Marking Block Signals” on page 9-10 for descriptions of how to mark block properties and block signals.

## Marking Block Parameters

Tagging parameters in your Simulink model allows the function `xpcsliface` to create To xPC Target interface blocks. These interface blocks contain the parameters you connect to control devices in your user interface model.

After you create a Simulink model, you can mark the block parameters. This procedure uses the model `xpctank.mdl` as an example.

---

**Tip** The `xpctank` model blocks and signals may contain placeholder tags illustrating the syntax. As you create your own copy of the model using these procedures, replace these tags with your new tags or add the new tags using the multiple label syntax.

---

- 1 Open a Simulink model. For example, in the MATLAB Command Window, type

xpctank

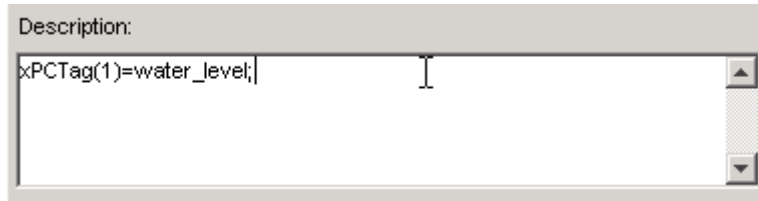
- 2 Point to a Simulink block, and then right-click.
- 3 From the menu, click **Block Properties**.



A Block properties dialog box opens.

- 4 In the **Description** box, delete the existing tag and enter a tag to the parameters for this block.

For example, the SetPoint block is a constant with a single parameter that selects the level of water in the tank. Enter the tag shown below.



The tag has the following format syntax

```
xPCTag(1, . . . index_n)= label_1 . . . label_n;
```

For single dimension ports, the following syntax is also valid:

```
xPCTag=label;
```

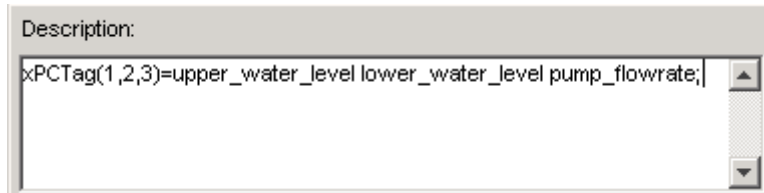
*index\_n* -- Index of a block parameter. Begin numbering parameters with an index of 1.

*label\_n* -- Name for a block parameter that will be connected to a To xPC Target block in the user interface model. Separate the labels with a space, not a comma.

label\_1...label\_n must consist of the same identifiers as those used by C/C++ to name functions, variables, and so forth. Do not use names like -foo.

- 5 Repeat steps 1 through 3 for the remaining parameters you want to tag.

For example, for the Controller block, enter the tag



For the PumpSwitch and ValveSwitch blocks, enter the following tags respectively:

```
xPCTag(2)=pump_switch;
```

```
xPCTag(1)=drain_valve;
```

To create the To xPC blocks in an user interface model for a block with four properties, use the following syntax:

```
xPCTag(1,2,3,4)=label_1label_2label_3label_4;
```

To create the To xPC blocks for the second and fourth properties in a block with at least four properties, use the following syntax:

```
xPCTag(2,4)=label_1 label_2;
```

- 6 From the **File** menu, click **Save as**. Enter a filename for your model. For example, enter

```
xpctank1
```

Your next task is to mark block signals if you have not already done so, and then create the user interface template model. See “Marking Block Signals” on page 9-10 and “Creating a Custom Graphical Interface” on page 9-3.

## Marking Block Signals

Tagging signals in your Simulink model allows the function `xpcsiface` to create From xPC Target interface blocks. These interface blocks contain the signals you connect to display devices in your user interface model.

After you create a Simulink model, you can mark the block signals. This procedure uses the model `xpctank1.mdl` (or `xpctank.mdl`) as an example. See “Creating a Target Application Model” on page 9-7.

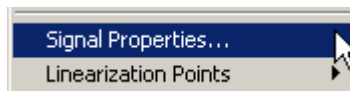
---

**Tip** The `xpctank` model blocks and signals may contain placeholder tags illustrating the syntax. As you create your own copy of the model using these procedures, replace these tags with your new tags or add the new tags using the multiple label syntax.

---

Note that you cannot select signals on the output ports of any virtual blocks such as Subsystem and Mux blocks. Also, you cannot select signals on any function-call, triggered signal output ports.

- 1 Open a Simulink model. For example, in the MATLAB Command Window, type  
`xpctank` or `xpctank1`
- 2 Point to a Simulink signal line, and then right-click.
- 3 From the menu, click **Signal Properties**.

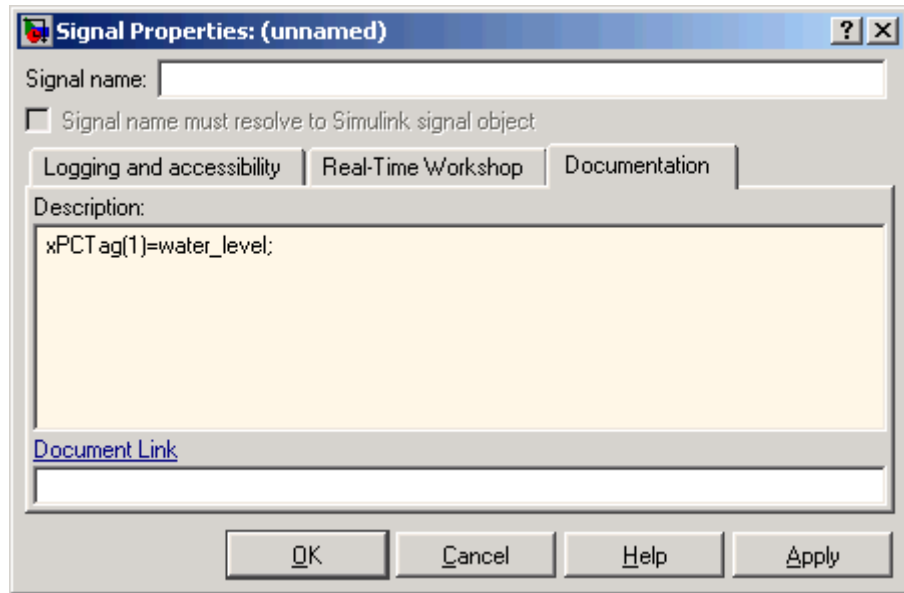


A Signal Properties dialog box opens.

- 4 Select the **Documentation** tab.
- 5 In the **Description** box, enter a tag to the signals for this line.



For example, the block labeled TankLevel is an integrator with a single signal that indicates the level of water in the tank. Replace the existing tag with the tag shown below.



The tag has the following format syntax:

```
xPCTag(1, . . . index_n)=label_1 . . . label_n;
```

For single dimension ports, the following syntax is also valid:

```
xPCTag=label;
```

- `index_n` — Index of a signal within a vector signal line. Begin numbering signals with an index of 1.
- `label_n` — Name for a signal that will be connected to a From xPC Target block in the user interface model. Separate the labels with a space, not a comma.

`label_1...label_n` must consist of the same identifiers as those used by C/C++ to name functions, variables, and so forth. Do not use names like `-foo`.

To create the From xPC blocks in an user interface model for a signal line with four signals (port dimension of 4), use the following syntax:

```
xPCTag(1,2,3,4)=label_1 label_2 label_3 label_4;
```

To create the From xPC blocks for the second and fourth signals in a signal line with at least four signals, use the following syntax:

```
xPCTag(2,4)=label_1 label_2;
```

---

**Note** Only tag signals from nonvirtual blocks. Virtual blocks are only graphical aids (see “Virtual Blocks”). For example, if your model combines two signals into the inputs of a Mux block, do not tag the signal from the output of the Mux block. Instead, tag the source signal from the output of the originating nonvirtual block.

---

- 6 From the **File** menu, click **Save as**. Enter a filename for your model. For example, enter

```
xpc_tank1
```

You next task is to mark block parameters if you have not already done so. See “Marking Block Parameters” on page 9-7. If you have already marked block signals, return to “Creating a Custom Graphical Interface” on page 9-3 for additional guidance on creating a user interface template model.

# Execution Using the Target Computer Command Line

---

You can interact with the xPC Target environment through the target computer command window. The xPC Target software provides a limited set of commands that you can use to work with the target application after it has been loaded to the target computer, and to interface with the scopes for that application.

## Target Computer Command-Line Interface

This interface is useful with standalone applications that are not connected to the host PC. You can type commands directly from a keyboard on the target computer. As you start to type at the keyboard, a command window appears on the target computer screen.

For a complete list of target computer commands, refer to “Target Computer Commands”

### In this section...

“Using Target Application Methods on the Target Computer” on page 10-2

“Manipulating Target Object Properties from the Target Computer” on page 10-3

“Manipulating Scope Objects from the Target Computer” on page 10-4

“Manipulating Scope Object Properties from the Target Computer” on page 10-6

“Aliasing with Variable Commands on the Target Computer” on page 10-6

## Using Target Application Methods on the Target Computer

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

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

- 1 On the target computer, press **C**.

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

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

```
start
```

- 3** To stop the application, type

```
stop
```

Once the command window is active, you do not have to reactivate it before typing the next command.

## Manipulating Target Object Properties from the Target Computer

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

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 computer.

- 1** On the target computer keyboard, press **C**.

The target computer activates the command window.

- 2** Type a target command. For example, to change the frequency of the signal generator (parameter 1) in the model `xpcosc`, type

```
setpar 1=30
```

The command window displays a message to indicate that the new parameter has registered.

```
System: p[1] is set to 30.00000
```

- 3** Check the value of parameter 1. For example, type

```
p1
```

The command window displays a message to indicate that the new parameter has registered.

```
System: p[1] is set to 30.00000
```

- 4 Check the value of signal 0. For example, type

```
s0
```

The command window displays a message to indicate that the new parameter has registered.

```
System: S0 has value 5.1851
```

- 5 Change the stop time. For example, to set the stop time to 1000, type

```
stoptime = 1000
```

The parameter changes are made to the target application but not to the target object. When you type any xPC Target command in the MATLAB Command Window, the target computer returns the current properties of the target object.

---

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

---

To see the correlation between a parameter or signal index and its block, you can look at the `model_name_pt.c` or `model_name_bio.c` of the generated code for your target application.

## Manipulating Scope Objects from the Target Computer

The xPC Target software uses a scope object to represent your target scope. This section shows some of the common tasks that you use with scope objects.

These commands create a temporary difference between the behavior of the target application and scope object. The next time you access the scope object, the data is updated from the target computer.

- 1 On the target computer keyboard, press **C**.

The target computer activates the command window.

- 2 Type a scope command. For example, to add a target scope (scope 2) in the model `xpcosc`, type

```
addscope 2
```

The xPC Target software adds another scope monitor to the target computer screen. The command window displays a message to indicate that the new scope has registered.

```
Scope: 2, created, type is target S0
```

- 3 Type a scope command. For example, to add a signal (0) to the new scope, type

```
addsignal 2=0
```

The command window displays a message to indicate that the new parameter has registered.

```
Scope: 2, signal 0 added
```

You can add more signals to the scope.

- 4 Type a scope command. For example, to start the scope 2, type

```
startscope 2
```

The target scope 2 starts and displays the signals you added in the previous step.

---

**Note** If you add a target scope from the target computer, you need to start that scope manually. If a target scope is in the model, starting the target application starts that scope automatically.

---

## Manipulating Scope Object Properties from the Target Computer

This section shows some of the common tasks that you use with target objects and their properties.

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 computer.

- 1 On the target computer keyboard, press **C**.

The target computer activates the command window.

- 2 Type a scope property command. For example, to change the number of samples (1000) to acquire in scope 2 of the model `xpcosc`, type

```
numsamples 2=1000
```

- 3 Type a scope property command. For example, to change the scope mode (numerical) of scope 2 of the model `xpcosc`, type

```
scopemode 2=numerical
```

The target scope 2 display changes to a numerical one.

## Aliasing with Variable Commands on the Target Computer

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

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

- 1 On the target computer keyboard, 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
```



The target computer command window is activated when you start to type, and a command line opens.

- 2** Type the variable name to run that command sequence. For example, to turn the motor on, type

on

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



# Execution Using the Web Browser Interface

---

## Web Browser Interface

In this section...
“Introduction” on page 11-2
“Connecting the Web Interface Through TCP/IP” on page 11-2
“Connecting the Web Interface Through RS-232” on page 11-3
“Using the Main Pane” on page 11-7
“Changing WWW Properties” on page 11-9
“Viewing Signals with a Web Browser” on page 11-10
“Viewing Parameters with a Web Browser” on page 11-11
“Changing Access Levels to the Web Browser” on page 11-11

### Introduction

The xPC Target software has a Web server that allows you to interact with your target application through a Web browser. You can access the Web browser with either a TCP/IP or serial (RS-232) connection.

---

**Note** RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.

---

The xPC Target Web server is built into the kernel that allows you to interact with your target application using a Web browser. If the target computer is connected to a network, you can use a Web browser to interact with the target application from any host computer connected to the network.

### Connecting the Web Interface Through TCP/IP

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

The TCP/IP stack on the xPC Target kernel supports only one simultaneous connection, because its main objective is real-time applications. This

connection is shared between the MATLAB interface and the Web browser. You must close any open connection to the target computer before you connect using the host computer Web browser. This also means that only one browser or the MATLAB interface is able to connect at one time.

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

---

**Note** Close all other connections to the target computer. For example, if you are currently connected to the target computer through xPC Target Explorer, right-click on that target computer icon and select **Disconnect**.

---

**1** In the MATLAB window, type

```
xpcwwenable
```

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

**2** Open a Web browser. In the address box, enter the IP address and port number you entered in the xPC Target Explorer window. For example, if the target computer IP address is 192.168.0.10 and the port is 22222, type

```
http://192.168.0.10:22222/
```

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

## Connecting the Web Interface Through RS-232

If the host computer and target computer are connected with a serial cable instead of a network cable, you can still connect the target application on the target computer to a Web browser on the host computer. The xPC Target software includes a TCP/IP to RS-232 mapping application. This application runs on the host computer and writes whatever it receives from the RS-232 connection to a TCP/IP port, and it writes whatever is received from the

TCP/IP port to the RS-232 connection. TCP/IP port numbers must be less than  $2^{16} = 65536$ .

Before you connect your Web browser on the host computer, you must load a target application onto the target computer. 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

```
xpcwwenable or close(xpc)
```

The MATLAB interface is disconnected from the target computer, leaving the target computer ready to connect to another client. The TCP/IP stack of the xPC Target kernel supports only one simultaneous connection. If you do not use this command, the TCP/IP to RS-232 gateway might not be able to connect to the target computer.

- 2** Open a DOS command window, and enter the command to start the TCP/IP to RS-232 gateway. For example, if the target computer is connected to COM1 and you would like to use the TCP/IP port 22222, type the following:

```
c:\<MATLAB root>\toolbox\rtw\targets\xpc\xpc\bin\xpctcp2ser  
-v -t 22222 -c 1
```

For a description of the xpctcp2ser command, see “Syntax for the xpctcp2ser Command” on page 11-5.

The TCP/IP to RS-232 gateway starts running, and the DOS command window displays the message

```
*-----*  
*           xPC Target TCP/IP to RS-232 gateway           *  
*           Copyright 2000 The MathWorks                 *  
*-----*  
Connecting COM to TCP port 22222  
Waiting to connect
```

If you did not close the MATLAB to target application connection, xpctcp2ser 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 panes.

- 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 RS-232 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
```

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

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

```
Error in ==>
C:\MATLABR13\toolbox\rtw\targets\xpc\xpc\@xpc\xpc.m
On line 31 ==> sync(xpcObj);
```

You must close the MATLAB interface and then restart it.

### Syntax for the `xpctcp2ser` Command

The `xpctcp2ser` command starts the TCP/IP to RS-232 gateway. The syntax for this command is

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

The options are described in the following table.

<b>Command-Line Option</b>	<b>Description</b>
-v	Verbose mode. Produces a line of output every time a client connects or disconnects.
-n	<p>Allows nonlocal 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.</p> <p>If you do not use this option, only the host computer that is connected to the target computer with a serial cable can connect to the selected port. For example, if you start the gateway on your host computer, with the default ports, you can type in the Web browser <code>http://localhost:2222</code>. However, if you try to connect to <code>http://Domainname.com:2222</code>, you will probably get a connection error.</p>
-t tcpPort	Use TCP port tcpPort. Default t is 22222. For example, to connect to port 20010, type -t 20010.
-h	Print a help message.
-c comPort	Use COM port comPort (1 <= comPort <= 4). Default is 1. For example, to use COM2, type -c 2.



## Using the Main Pane

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

The screenshot displays the 'Main Pane' of a web browser interface, which is divided into four distinct sections. To the right of these sections is a large, light blue area containing the instruction: "Click any button on the left to navigate".

**System Status**

Application	xprosc
Mode	Real-Time Single-Tasking
Status	Stopped
CPUOverload	none
ExecTime	0.0
SessionTime	88641.1
StopTime	0.2
SampleTime	0.00025
AvgTET	1.04013e-005

Start Execution

Get State Log

Get Output Log

Get TET Log

**xPC Target Properties**

ViewMode: All

SampleTime: 0.00025

StopTime: 0.2

Apply    Reset

**Navigation**

Scopes

Signals

Parameters

Screen Shot

Refresh

**WWW Properties**

Maximum Signal Width: Inf

Refresh Interval: /

After you connect a Web browser to the target computer, you can use the **Main** pane 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 computer. If the right frame is either the **Signals List** pane or the **Screen Shot** pane, updating the left frame also updates the right frame.

System Status	
Application	<b>xpcosc</b>
Mode	<b>Real-Time Single-Tasking</b>
Status	<b>Stopped</b>
CPUOverload	<b>none</b>
ExecTime	<b>0.0</b>
SessionTime	<b>97769</b>
StopTime	<b>10000</b>
SampleTime	<b>0.00025</b>
AvgTET	<b>-nan</b>

- 2 Click the **Start Execution** button.

The target application begins running on the target computer, 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 **StopTime** and **SampleTime** boxes, then click the **Apply** button. You can enter -1 or Inf in the **StopTime** box for an infinite stop time.

SampleTime	<input type="text" value="0.00025"/>
StopTime	<input type="text" value="10000"/>
<input type="button" value="Apply"/>	<input type="button" value="Reset"/>

The new property values are downloaded to the target application. Note that the **SampleTime** box is visible only when the target application is stopped. You cannot change the sample time while a target application is running. (See “User Interaction” in the *xPC Target Getting Started Guide* for limitations on changing sample times.)

- 5** Select scopes to view on the target computer. From the **ViewMode** list, select one or all of the scopes to view.

ViewMode	<input type="text" value="All"/>
	<input type="text" value="All"/> <input checked="" type="text" value="Scope 1"/> <input type="text" value="Scope 3"/>

---

**Note** The **ViewMode** control is visible in the **xPC Target Properties** pane only if you add two or more scopes to the target computer.

---

## 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** box 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 or equal to n).

Signals with a width greater than the value you enter are not displayed on the **Signals** pane.

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

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

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

This can also happen when you are trying to update a parameter or property at the same time that the pane is automatically refreshing.

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

### Viewing Signals with a Web Browser

The **Signals** pane is a list of the signals in your model.

After you connect a Web browser to the target computer you can use the **Signals** pane to view signal data:

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

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

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

The **Signals** pane is updated with the current values. Vector/matrix signals are expanded and indexed in the same column-major format that the MATLAB interface uses. This can 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** pane is loaded and a copy of the current target computer screen is displayed. The screen shot uses the portable network graphics (PNG) file format.

## Viewing Parameters with a Web Browser

The **Parameters** pane 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 computer, you can use the **Parameters** pane 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** pane is loaded into the right frame.

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

If the parameter is a vector or matrix, click the **Edit** button to view the vector or matrix. You can edit the parameter in this pane.

- 2 In the **Value** box, enter a new parameter value, and then click the **Apply** button.

## 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 **Configuration Parameters**.

The Configuration Parameters dialog box for the model is displayed.

- 2 Click the **Code Generation** node.

The code generation pane opens.

**3** In the **Target selection** section, 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 highest access level (0) is set.

**4** 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 do not see the button for parameters.

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

**Level 0** — Full access to all panes 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 or log data.

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

**Level 4** — Cannot edit existing scopes on the **Scopes** pane. Cannot add or remove signals on the **Scopes** pane. Cannot view the **Signals** pane and the **Parameters** pane, and cannot get scope data.

# Troubleshooting

---

Refer to these guidelines, hints, and tips for questions or issues you might have about your installation of the xPC Target product. For more specific troubleshooting solutions, go to the MathWorks Support xPC Target Web site (<http://www.mathworks.com/support/product/XP>) for specific troubleshooting solutions.

- Chapter 12, “Basic Troubleshooting”
- Chapter 13, “Confidence Test Failures”
- Chapter 14, “Host Computer Configuration”
- Chapter 15, “Target Computer Configuration”
- Chapter 16, “Host-Target Communication”
- Chapter 17, “Target Computer Boot Process”
- Chapter 18, “Modeling”
- Chapter 19, “Model Compilation”
- Chapter 20, “Application Download”
- Chapter 21, “Application Execution”
- Chapter 22, “Application Parameters”
- Chapter 23, “Application Signals”
- Chapter 24, “Application Performance”
- Chapter 25, “Getting MathWorks Support”





# Basic Troubleshooting

---

## Basic Troubleshooting

An xPC Target installation can sometimes fail. Causes include hardware failures, changes in underlying system software, and procedural errors. Follow this procedure to address these problems:

- 1 Run the confidence test (see “Running the Confidence Test”).

---

**Tip** Run the confidence test as the first step in troubleshooting, as well as in validating your initial product installation and configuration.

---

- 2 If any tests fail, see the following information about the specific failure:
  - “Test 1, Ping target PC 'TargetPC1' using system ping: FAILED” on page 13-2
  - “Test 2, Ping target PC 'TargetPC1' using xpctargetping: FAILED” on page 13-5
  - “Test 3, Software reboot the target PC: FAILED” on page 13-7
  - “Test 4, Build and download an xPC Target application using model xpcosc: FAILED” on page 13-9
  - “Test 5, Check host-target command communications: FAILED” on page 13-12
  - “Test 6, Download a pre-built xPC Target application: FAILED” on page 13-14
  - “Test 7, Execute xPC Target application for 0.2s: FAILED” on page 13-15
  - “Test 8, Upload logged data and compare with simulation results: FAILED” on page 13-16
- 3 Check the categorized questions and answers for clues to the root cause of the problem.
- 4 If the tests run, but test execution time is slow or the CPU overloads, see Chapter 24, “Application Performance”.

- 5** Check the MathWorks Support web site and MATLAB Central for tips. See “How Do I Find the MathWorks Support Web Site?” on page 25-2.
- 6** Call MathWorks Technical Support. See “How Do I Contact MathWorks Technical Support?” on page 25-5.



# Confidence Test Failures

---

This topic describes guidelines, hints, and tips for questions or issues you might have while using the xPC Target product. Refer to the MathWorks Support xPC Target Web site (<http://www.mathworks.com/support/product/XP>) for specific troubleshooting solutions. The xPC Target documentation is also available from this site.

- “Test 1, Ping target PC 'TargetPC1' using system ping: FAILED” on page 13-2
- “Test 2, Ping target PC 'TargetPC1' using xpctargetping: FAILED” on page 13-5
- “Test 3, Software reboot the target PC: FAILED” on page 13-7
- “Test 4, Build and download an xPC Target application using model xpcosc: FAILED” on page 13-9
- “Test 5, Check host-target command communications: FAILED” on page 13-12
- “Test 6, Download a pre-built xPC Target application: FAILED” on page 13-14
- “Test 7, Execute xPC Target application for 0.2s: FAILED” on page 13-15
- “Test 8, Upload logged data and compare with simulation results: FAILED” on page 13-16

## Test 1, Ping target PC 'TargetPC1' using system ping: FAILED

If you are using a network connection, this test is a standard system ping to your target computer.

---

### Note

- The confidence test skips test 1 for serial connections.
  - xPC Target Explorer runs only on 32-bit host computers. On 64-bit computers, use the MATLAB command-line interface. For further information, see “Configuring Environment From the MATLAB Command Line” on page 4-21.
- 

Troubleshoot failures with the following procedure:

- 1** Open a DOS shell and type the IP address of the target computer:

```
ping xxx.xxx.xxx.xxx
```

Check the messages on your screen.

If DOS displays a message similar to the following, system ping succeeds even though test 1 fails.

```
Pinging xxx.xxx.xxx.xxx with 32 bytes of data:  
Reply from xxx.xxx.xxx.xxx: bytes=32 time<10 ms TTL=59
```

If the DOS shell displays the following message, the system ping command failed.

```
Pinging xxx.xxx.xxx.xxx with 32 byte of data:  
Request timed out.
```

- 2** Ping succeeds — Ethernet addresses OK?

If ping succeeds, check whether you entered the required IP and gateway addresses in xPC Target Explorer:

- a In the MATLAB Command Window, type  

```
xpcexplr
```
- b Select the Configuration Communication dialog for the failing target configuration.
- c Check that **Target PC IP address**, **LAN subnet mask address**, and **TCP/IP gateway address** have the required values.
- d Change the TCP/IP options as required.
- e Create a new network boot image or boot drive and reboot the target computer with the new image. See “Target Boot Method”.

### 3 Ping fails — Cables OK?

If ping fails, first check your network cables. You might have a faulty network cable or, if you are using a coaxial cable, the terminators might be missing.

### 4 Ping fails — xPC Target properties OK?

Check that you have entered all required properties in xPC Target Explorer.

To solve this problem:

- a In the MATLAB Command Window, type  

```
xpcexplr
```
- b Select the Configuration Communication dialog for the failing target configuration.
- c Check that **Target PC IP address**, **LAN subnet mask address**, and **TCP/IP gateway address** have the required values.
  - For a PCI bus, check that **TCP/IP target bus type** is set to PCI instead of ISA.
  - For an ISA bus:
    - Check that **TCP/IP target bus type** is set to ISA instead of PCI.

- Check that **TCP/IP target ISA memory port** is set to the required I/O port base address and check that the address does not lead to a conflict with another hardware resource.
  - Check that **TCP/IP target ISA IRQ number** is set to the required IRQ line and check that the line number does not lead to a conflict with another hardware resource.
  - If the target computer motherboard contains a PCI chip set, check whether the IRQ line used by the ISA bus Ethernet card is reserved within the BIOS setup.
- d** Change the TCP/IP options as required.
- e** Create a new network boot image or boot drive and reboot the target computer with the new image. See “Target Boot Method”.

### **5 Ping fails — Ethernet hardware operating?**

Verify that your hardware is operating. For example, check that the green “ready” light goes on when the cable is connected to the Ethernet card.

### **6 Ping fails — Ethernet card supported?**

Verify that you are using a supported Ethernet card on the target computer. See “Hardware for Network Communication” of the *xPC Target Getting Started Guide* for further details, including supplied Ethernet cards.

### **7 Ping fails — Not a locally mounted folder?**

Run `xpctest` from a locally mounted folder, such as `Z:\work`, rather than from a UNC network folder, such as `\\Server\user\work`.

### **8** If these steps do not solve your problem, check the following:

- Chapter 16, “Host-Target Communication”
- “Is the Target Computer BIOS Set as Required?” on page 15-2

### **9** If you still cannot solve your problem, see Chapter 25, “Getting MathWorks Support”.



## Test 2, Ping target PC 'TargetPC1' using xpctargetping: FAILED

This test is an xPC Target ping to your target computer. Troubleshoot failures with the following procedure:

- 1 In the MATLAB Command Window, type

```
tg=xpctarget.xpc('argument-list')
```

where `argument-list` is the connection information that indicates which target computer you are working with. If you do not specify any arguments, the software assumes that you are communicating with the default target computer.

Check the messages in the MATLAB Command Window.

MATLAB should respond with the following messages:

```
xPC Object
  Connected          = Yes
  Application        = loader
```

### 2 Not connected — Bad target boot drive?

If you do not get the preceding messages, you could have a bad target boot drive. To solve this problem, create a new network boot image or boot drive and reboot the target computer with the new image. See “Target Boot Method”.

### 3 Not connected — Environment variables set?

Use the PC MATLAB command to check the environment variables, in particular **Target PC IP address**. If test 1 passes but test 2 fails, you might not have entered the required IP address.

### 4 Not connected — Ethernet card supported?

If you are using a TCP/IP connection, make sure you are using a supported Ethernet card (see “Test 1, Ping target PC 'TargetPC1' using system ping: FAILED” on page 13-2).

**5 Not connected — RS-232 configuration?**

If you are using an RS-232 connection, check the following:

- Verify that you are using a null modem cable (see “Hardware for Serial Communication”).
- Verify that the COM ports on the host and target computers are enabled in the BIOS. If they are disabled, test 2 fails.
- Verify that the specified COM port is connected on each computer.
- Verify that the COM port being used matches the port specified in the target computer configuration.

---

**Note** RS-232 host-target communication mode will be removed in a future release. Use TCP/IP instead.

---

**6** If these steps do not solve your problem, check the following:

- Chapter 16, “Host-Target Communication”
- “Is the Target Computer BIOS Set as Required?” on page 15-2

**7** If you still cannot solve your problem, see Chapter 25, “Getting MathWorks Support”.

## Test 3, Software reboot the target PC: FAILED

This test tries to boot your target computer using an xPC Target command.

---

**Note** This procedure assumes that you have set environment settings with xPC Target Explorer. See “Environment Properties for Serial Communication” or “Environment Properties for Network Communication”.

---

Troubleshoot failures with the following procedure:

- 1 In the MATLAB Command Window, type

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

This command reruns the test without using the `xpctarget.xpc.reboot` command and displays the message

```
### Test 3, Software reboot the target PC: ... SKIPPED
```

### 2 Build Succeeded — Software reboot supported?

Check the results of Test 4, Build and download an xPC Target application using model `xpcosc`. If `xpctest` skips the `xpctarget.xpc.reboot` command but builds and loads the target application without an error, the problem could be that the target computer does not support the xPC Target reboot command. In this case, you need to reboot using a hardware reset button.

### 3 Build Failed — Kernel not loaded?

If you saw the following error, the kernel might not be loaded when the host computer initiates communication with the target computer.

```
ReadFile Error: 6
```

Older xPC Target releases might receive this error. As a workaround, run `xpctest` with the `noreboot` option. For example,

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

This command runs the test without trying to reboot the target computer. It displays the following message:

```
### Test 3, Software reboot the target PC: ... SKIPPED
```

#### **4 Build Failed — Demo model modified?**

If you directly or indirectly modify the `xpcosc` demo model supplied with the product, test 3 is likely to fail.

---

**Note** Do not modify any of the files installed with the xPC Target software. If you want to modify one of these files, copy the file and modify the copy.

---

Restore the `xpcosc` demo model to its original state by one of the following methods:

- Recreate the original model by editing it in the following location:

```
matlabroot\toolbox\rtw\targets\xpc\xpcdemos
```

- Reinstall the software.

#### **5** If these steps do not solve your problem, check the following:

- Chapter 17, “Target Computer Boot Process”
- “Is the Target Computer BIOS Set as Required?” on page 15-2

#### **6** If you still cannot solve your problem, see Chapter 25, “Getting MathWorks Support”.

## Test 4, Build and download an xPC Target application using model xpcosc: FAILED

This test tries to build and download the model `xpcosc.mdl`.

---

**Note** xPC Target Explorer runs only on 32-bit host computers. On 64-bit computers, use the MATLAB command-line interface. For further information, see “Configuring Environment From the MATLAB Command Line” on page 4-21.

---

Troubleshoot failures with the following procedure:

- 1 In the MATLAB Command Window, check the error messages.

These messages help you locate where there is a problem.

- 2 **Build Failed — Loader not ready?**

If you get the following error message, reboot your target computer:

```
xPC Target loader not ready
```

This error message is sometimes displayed even if the target screen shows the loader is ready.

- 3 **Build Failed — Using full duplex?**

If the communication between the host computer and target computer is TCP/IP, set the host computer network interface card (NIC) card and hub to half-duplex mode. Do not set the mode to full-duplex mode.

- 4 **Build Failed — Compiler not supported?**

Verify that a supported compiler is being used and that all of the blocks in the model can be compiled with the given compiler and compiler version.

- 5 **Build Failed — Compiler path?**

Verify the specified path to the supported compiler. You need only the root path to the compiler, not the full path. If you did not enter the specified path, you might get one of the following errors:

```
Error executing build command: Error using ==> make_rtw
Error using ==> rtw_c (SetupForVisual)
Invalid DEVSTUDIO path specified
```

or

```
Error executing build command: Error using ==> make_rtw
Error using ==> rtw_c
Errors encountered while building model "xpcosc"
```

with the following MATLAB Command Window error:

```
NMAKE: fatal error U1064: MAKEFILE not found and no target
specified
Stop.
```

- a** Verify that your compiler is installed in the expected location. For example, all Microsoft Visual compiler components must be in the Microsoft Visual Studio® folder after installation.
- b** At the MATLAB prompt, type

```
xpcexplr
```
- c** In the **Select C compiler** field, select the compiler type corresponding to your compiler (VisualC or Watcom).

---

**Note** WATCOM compiler support will be removed in a future release. Use a Microsoft compiler instead.

---

- d** In the **Compiler Path** field, enter the root path to the compiler. For example, on a 32-bit computer, enter:

```
c:\Program Files\Microsoft Visual Studio 10.0
```

On a 64-bit computer, enter:

c:\Program Files (x86)\Microsoft Visual Studio 10.0

Do not add a terminating back slash (\) at the end of the path.

## **6 Build Failed — COM port read failed?**

If you see the following MATLAB Command Window error:

ReadFile failed while reading from COM-port

- Check the state of your target computer. If it is unresponsive, you might need to reboot the target computer.
- In the xPC Target Explorer, try to connect to the target computer again. Be sure to also check the connection between the host computer and target computer.

**7** If these steps do not solve your problem, check the following:

- Chapter 19, “Model Compilation”
- Chapter 20, “Application Download”
- Chapter 16, “Host-Target Communication”
- “Is the Target Computer BIOS Set as Required?” on page 15-2

**8** If you still cannot solve your problem, see Chapter 25, “Getting MathWorks Support”.

## Test 5, Check host-target command communications: FAILED

This error occurs only when the environment variable settings are out of date.

---

**Note** xPC Target Explorer runs only on 32-bit host computers. On 64-bit computers, use the MATLAB command-line interface. For further information, see “Configuring Environment From the MATLAB Command Line” on page 4-21.

---

Troubleshoot failures with the following procedure:

- 1 At the MATLAB prompt, start xPC Target Explorer. For example,  

```
xpcexplr
```
- 2 Verify the environment variables for the target computer.
- 3 If you have xPC Target Embedded Option installed, verify that, in the Configuration section, you have selected the **Standalone** tab.
- 4 Make the required changes.
- 5 Select the tab for your boot mode. For example, **CD Boot**.

---

**Tip** For information on boot options, see “Target Boot Method” in the *xPC Target Getting Started Guide*.

---

- 6 Recreate the network boot image or boot drive.
- 7 Reboot the target computer.
- 8 Rerun `xpctest`.
- 9 If this does not resolve the issue, recreate the boot drive or image using `xpcbootdisk`, reboot the target computer, and rerun `xpctest`.



- 10** If these steps do not solve your problem, check the following:
- Chapter 16, “Host-Target Communication”
  - “Is the Target Computer BIOS Set as Required?” on page 15-2
- 11** If you still cannot solve your problem, see Chapter 25, “Getting MathWorks Support”.

## **Test 6, Download a pre-built xPC Target application: FAILED**

This test runs the basic target object constructor, `xpc`. This error rarely occurs unless an earlier test has failed.

- 1** Verify that all preceding steps completed with no error.
- 2** Run the tutorial model (see “Tutorial and Examples”) and record any error messages that result.
- 3** Check the following as suggested by the tutorial model error messages:
  - Chapter 20, “Application Download”
  - Chapter 16, “Host-Target Communication”
  - “Is the Target Computer BIOS Set as Required?” on page 15-2
- 4** If you still cannot solve your problem, see Chapter 25, “Getting MathWorks Support”.

## Test 7, Execute xPC Target application for 0.2s: FAILED

This test executes a target application (xpcosc) on the target computer. This test fails if you change the xpcosc model start time to something other than 0, such as 0.001. This change causes the test, and the MATLAB interface, to halt. To troubleshoot this failure:

- 1** Set the xpcosc model start time back to 0.
- 2** Rerun the test.
- 3** If these steps do not solve your problem, check the following:
  - Chapter 21, “Application Execution”
  - Chapter 24, “Application Performance”
  - Chapter 23, “Application Signals”
  - Chapter 22, “Application Parameters”
  - “Is the Target Computer BIOS Set as Required?” on page 15-2
- 4** If you still cannot solve your problem, see Chapter 25, “Getting MathWorks Support”.

## Test 8, Upload logged data and compare with simulation results: FAILED

This test executes a target application (xpcosc) on the target computer. This test might fail if you change the xpcosc model (for example, if you remove the Outport block).

---

**Note** Do not modify any of the files installed with the xPC Target software. If you want to modify one of these files, copy the file and modify the copy.

---

- 1 To eliminate this problem, restore the xpcosc demo model to its original state by one of the following methods:
  - Recreate the original model by editing it in the following location:  

```
matlabroot\toolbox\rtw\targets\xpc\xpcdemos
```
  - Reinstall the software.
- 2 Other issues might also cause this test to fail. If you still need more help, check the following:
  - If you are running a new xPC Target release, be sure that you have a new boot drive or image for this release. See “What Should I Do After I Get a New Release?” on page 25-4.
  - There is a known issue with xPC Target software version 1.3. It might occur when you run `xpctest` two consecutive times. See the known issue and solution documented in <http://www.mathworks.com/support/solutions/data/1-18DTB.html>.
- 3 If you are installing another version of the xPC Target software on top of an existing version, check the version number of the current installation. At the MATLAB command line, type `xpclib`. The version number appears at the bottom of the xPC Target block library window. If the version number is not the one to which you want to upgrade, reinstall the software.
- 4 If these steps do not solve your problem, check the following:
  - Chapter 21, “Application Execution”

- Chapter 24, “Application Performance”
  - Chapter 23, “Application Signals”
  - Chapter 22, “Application Parameters”
  - “Is the Target Computer BIOS Set as Required?” on page 15-2
- 5** If you still cannot solve your problem, see Chapter 25, “Getting MathWorks Support”.



# Host Computer Configuration

---

### Why Does Boot Drive Creation Halt Without Finishing?

If your host computer MATLAB interface halts while creating an xPC Target boot disk or network boot image:

- Use another drive to create a new xPC Target boot drive or network boot image.
- If your host computer has antivirus software, it might conflict with the MATLAB software. Disable the software while using the MATLAB interface.
- Verify that the host computer drive is accessible. If it is not accessible, replace the drive.



# Target Computer Configuration

---

- “Is the Target Computer BIOS Set as Required?” on page 15-2
- “Can the Target Computer Hard Drive Contain Multiple Partitions?” on page 15-3
- “Why Do I Get a File System Disabled Error?” on page 15-4
- “How Can I Adjust the Stack Size on My Target Computer?” on page 15-5
- “How Can I Obtain PCI Board Information for My Target Computer?” on page 15-6
- “What Do I Do If My I/O Board Does Not Work?” on page 15-8

## Is the Target Computer BIOS Set as Required?

The BIOS settings of a computer system can affect how the computer works. If you experience problems using the xPC Target software, you should check the system BIOS settings of the target computer. These settings are beyond the control of the xPC Target product. See “Target Computer BIOS Settings” in the *xPC Target Getting Started Guide*.

Faulty BIOS settings can cause issues like the following:

- Why is my target not booting?
- Why can `getxpcpci` detect PCI boards, but `autosearch -1` cannot?
- Why can my standalone application run on some target computers, but not others?
- Why is my target computer crashing while downloading applications?
- Why is my target PC104 hanging on boot?
- Why is my boot time slow?
- Why is my software not running in real time?
- Why are my USB ports not working?

## **Can the Target Computer Hard Drive Contain Multiple Partitions?**

Yes, the target computer hard drive can contain multiple partitions. However, the xPC Target software supports file systems of type FAT-12, FAT-16, or FAT-32 only.

## Why Do I Get a File System Disabled Error?

If your target computer does not have a FAT hard disk, the monitor on the target computer displays the following error:

```
ERROR -4: drive not found  
No accessible disk found: file system disabled
```

If you do not want to access the target computer file system, you can ignore this message. If you want to access the target computer file system, add a FAT hard disk to the target computer system and reboot.

---

**Tip** Verify that the hard drive is not cable-selected and that the BIOS can detect it.

---

## How Can I Adjust the Stack Size on My Target Computer?

To discover and adjust the stack size used by the real-time threads on the target computer:

**1** Add the following blocks to your model:

- xPC Target Get Free Stack Size — Outputs the number of bytes of stack memory currently available to the target application thread.
- xPC Target Get Minimal Free Stack Size — Outputs the number of bytes that have never been used in the stack since the thread was created.

---

**Note** The underlying function traverses the entire stack to find unused bytes. For performance reasons, Get Minimal Free Stack Size should be used only for diagnostic purposes.

---

**2** Execute the target application, monitoring the stack size and minimal stack size.

**3** Calculate a stack size that allows execution to proceed.

---

### Note

- To meet the memory requirements, you might have to reconfigure your target computer.
  - The xPC Target kernel can use only 2 GB of memory.
- 

**4** Adjust the stack size of the real-time threads by setting a TLC option in the Configuration Parameters dialog, Code Generation node, section Build Process.

For example, to set the stack size to 256 kBytes, type the following in the TLC option box:

```
-axPCModelStackSizeKB=256
```

## How Can I Obtain PCI Board Information for My Target Computer?

Information about the PCI devices in your target computer is useful if you want to determine what PCI boards are installed in your xPC Target system, or if you have multiple boards of a particular type in your system. Before you start, determine what boards are installed in your xPC Target system. Use one of the following:

- In the xPC Target Explorer, connect to the target computer in question and expand the **PCI Devices** node.
- In the MATLAB Command Window, type

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

If you have or want to use multiple boards of a particular type in your system, verify that the I/O drive supports multiple boards. Refer to one of the following:

- xPC Target I/O Reference
- xPC Target Interactive Hardware Selection Guide  
([http://www.mathworks.com/support/product/XP/productnews/interactive\\_guide/xPC\\_Target\\_Interactive\\_Guide.html](http://www.mathworks.com/support/product/XP/productnews/interactive_guide/xPC_Target_Interactive_Guide.html))

If you confirm that the board type supports multiple boards, and these boards are installed in the xPC Target system, do the following to obtain the bus and slot information for these boards:

- 1 In the PCI devices display, note the contents of the **Bus** and **Slot** columns of the PCI devices in which you are interested.
- 2 Enter the bus and slot numbers as vectors into the **PCI Slot** parameter of the PCI device. For example:

```
[1 9]
```

where 1 is the bus number and 9 is the slot number.

For additional information about PCI bus I/O devices, refer to the “PCI Bus I/O Devices” section of xPC Target I/O Reference.

## What Do I Do If My I/O Board Does Not Work?

If you encounter issues using the xPC Target I/O drivers:

- 1** Display the input/output behavior of the board using an external instrument, such as an oscilloscope or logic analyzer.
- 2** Verify that you have configured the I/O board driver according to the manufacturer's data sheet.
- 3** Verify that you are using the latest version of the I/O board driver and of the xPC Target software. See "How Do I Get an Updated Software Release?" on page 25-3.
- 4** Verify that the behavior persists when you run the target application on a different target computer.
- 5** Verify that the behavior persists when you install another instance of the I/O board in the target computer.
- 6** Download the manufacturer's I/O driver and diagnostic software from the manufacturer web site, install the driver and software on your computer, and test the hardware using the manufacturer's software.
- 7** Report the issue to MathWorks Support at [http://www.mathworks.com/support/contact\\_us/index.html](http://www.mathworks.com/support/contact_us/index.html).



# Host-Target Communication

---

- “Is There Communication Between Your Computers?” on page 16-2
- “Are I/O Boards with Slow Initialization Times Causing Communication Problems During Download?” on page 16-4
- “Are Multiple Ethernet Cards in the Target Computer Causing Communication Problems During Download?” on page 16-6
- “Are Errors in I/O Board Drivers Causing Communication Problems During Download?” on page 16-8
- “How Can I Diagnose Network Problems?” on page 16-9

## Is There Communication Between Your Computers?

Use the following MATLAB commands from the host computer to validate the host/target setup:

- `xpctargetping`

The `xpctargetping` command performs a basic communication check between the host and target computers. This command returns **success** only if the xPC Target kernel is loaded and running and the host and target computer are communicating. Use this command for a quick check of the communication between the host computer and target computer.

- `xpctest`

The `xpctest` command performs a series of tests on your xPC Target system. These tests range from performing a basic communication check to building and running target applications. At the end of each test, the command returns an OK or failure message. If the test is inappropriate for your setup, the command returns a **SKIPPED** message. Use this command for a thorough check of your xPC Target installation.

Communication errors might also occur in the following instances:

- The target computer is running an old xPC Target boot drive or boot image that is not in sync with the xPC Target release installed on the host computer. Create a new boot drive or image for each new release.
- If the communication between the host computer and target computer is TCP/IP, set the host computer network interface card (NIC) card and hub to half-duplex mode. Do not set the mode to full-duplex mode.
- If you have an active firewall in your system, you might experience communication errors. For example, build errors might occur if you try to build and download a model with a thermocouple board (causing a slower initialization time) in a system that contains a firewall. To work around this issue, you can add the MATLAB interface to the firewall exception list. See also “Are I/O Boards with Slow Initialization Times Causing Communication Problems During Download?” on page 16-4
- To diagnose BIOS problems, see:
  - “Is the Target Computer BIOS Set as Required?” on page 15-2

- “Target Computer BIOS Settings”
- If multiple Ethernet cards or chips are installed in the target computer, see “Are Multiple Ethernet Cards in the Target Computer Causing Communication Problems During Download?” on page 16-6.

## Are I/O Boards with Slow Initialization Times Causing Communication Problems During Download?

Some xPC Target boards take a long time to initialize. This situation might cause the software to run out of time before a model downloads, causing the host computer to disconnect from the target computer.

By default, if the host computer does not get a response from the target computer after downloading a target application and waiting 5 seconds, the host computer software times out. The target computer responds only after downloading and initializing the target application.

Usually 5 seconds is enough time to initialize a target application, but in some cases it might not be long enough. The time to download a target application mostly depends on your I/O hardware. For example, thermocouple hardware (such as the PCI-DAS-TC board) takes longer to initialize. With slower hardware, you might also get errors when building and downloading an associated model. Even though the target computer is fine, a false timeout is reported and you might get an error like the following:

```
"cannot connect to ping socket"
```

This is not a fatal error. You can reestablish communication with the following procedure:

- 1 At the MATLAB Command Window, issue the `xpctargetping` command.  
For example:

```
xpctargetping
```

- 2 Wait for the system to return from the `xpctargetping`.
- 3 Restart the target object.

Alternatively, you can increase the timeout value, using the following procedure:

- 1 In your Simulink model, select **Simulation > Configuration Parameters > xPC Target options** and create a file called `xpcdltimeout.dat`.

**2** Clear the **Use default communication timeout** parameter.

The **Specify the communication timeout in seconds** parameter appears.

**3** Specify a new timeout value, in seconds. For example, enter 20.

**4** Click **OK**.

**5** Rebuild the model.

In this case, the host computer waits for about 20 seconds before declaring that a timeout has occurred. It does not take 20 seconds for every download. The host computer polls the target computer about once every second, and if a response is returned, returns the success value. Only in the case where a download really fails does it take the full 20 seconds.

## Are Multiple Ethernet Cards in the Target Computer Causing Communication Problems During Download?

The xPC Target product supports a number of Ethernet cards and chips, as described in “Hardware for Network Communication” in the *xPC Target Getting Started Guide*. If your target computer has more than one of these cards or chips installed, you could experience timeout problems. For example, suppose you are using the Network Boot option to boot the target computer. If the host computer boots the target computer using Ethernet A on the target computer, it associates the IP address of the target computer with the Media Access Control (MAC) address of Ethernet adapter A. If, after it does so, the target computer BIOS connects the target computer to Ethernet B, the xPC Target software cannot connect the host and target computers because they are connected to different Ethernet controllers.

First, try to disable or remove the Ethernet controller that you will not use. For example, if you have both an on-board Ethernet controller and a separate Ethernet card, you could disable the on-board Ethernet controller through the target computer BIOS. If you are required to have multiple Ethernet adapters of the same type in the target computer, you might need to experiment to determine which Ethernet adapter the software has chosen.

If you are not using the Network Boot option to boot the target computer and cannot establish communication between the target computer and host computer:

- 1 Switch the network cable to the other Ethernet port and try again.
- 2 If you can establish communication, use this Ethernet port to connect the host computer to the target computer.

If you are using the Network Boot option and experience this issue, do the following:

- 1 Connect the network cable to Ethernet adapter B.
- 2 In the MATLAB Command Window, type

```
!arp -d
```

This command removes the association between the target computer address and the hardware address of Ethernet adapter A from the cache of the host computer. This removal allows a new connection (and association) to be made.

**3** Change the Ethernet adapter card that the Network Boot option uses. You can do this in one of the following ways:

- Change the target computer BIOS to change the Ethernet adapter to the one that the Network Boot option is looking for.
- Set the `EthernetIndex` property. If the target computer is not the default, issue a series of commands like the following. These commands assume that the target computer name is `nonDefaultTarget`.

```
allTargets = xpctarget.targets;  
myTargetEnv = allTargets.Item('nonDefaultTarget');  
set(myTargetEnv, 'EthernetIndex', '1');
```

Recreate the Network Boot option boot image.

Reboot the target computer.

- Alternatively, if your system has a single target computer, you can use the `setxpcenv` command. In the MATLAB Command Window, type:

```
setxpcenv('EthernetIndex', '1')
```

Recreate the Network Boot option boot image.

Reboot the target computer.

## Are Errors in I/O Board Drivers Causing Communication Problems During Download?

If an error in a driver causes the xPC Target system to crash, a timeout occurs and `xpctargetping` fails with an error message. In such an event, you need to reboot the target and reestablish communication between the host computer and target computer.

To get the xPC Target system back up and running:

- 1** Remove the reference to the problem driver from the model.
- 2** Reboot the target computer.
- 3** At the MATLAB command line, issue `xpctargetping` to reestablish communications.
- 4** If the driver with which you are having problems is one provided by MathWorks, try to pinpoint the problem area (for example, determine whether certain settings in the driver block cause problems).

Alternatively, you can exit and restart the MATLAB interface.



## **How Can I Diagnose Network Problems?**

If you experience network problems when using this product, use an available computer with Internet access to refer to the MathWorks Support xPC Target Web site (<http://www.mathworks.com/support/product/XP>). This Web site has the most up-to-date information about this topic.



# Target Computer Boot Process

---

- “Why Is the Target Computer Unable to Boot?” on page 17-2
- “How Do I Fix a Kernel Load Error?” on page 17-4
- “How Do I Fix a Not Bootable Medium Error?” on page 17-5
- “Why Is the Target Computer Halted?” on page 17-6

## Why Is the Target Computer Unable to Boot?

If your target computer cannot boot with the xPC Target boot drive or network boot image:

- Use another disk, CD, or other medium and create a new xPC Target boot drive or network boot image.
- Verify that the current properties on the xPC Target boot drive correspond to the environment variables of xPC Target Explorer.
- Verify that the xPC Target boot drive contains files like the following:
  - BOOTSECT.RTT
  - XPCTGB1.RTA

---

**Note** The name of the last file varies depending on the communication method.

---

- If any of these files are not present, reinstall the software to fix any corrupted files from the previous installation.
- The xPC Target kernel may not be able to discover system hardware not compliant with the Advanced Configuration and Power Interface (ACPI) standard. To allow the kernel to discover such hardware, use the following xPC Target environment property to access the legacy MPFPS in the computer BIOS:

```
setxpcenv('LegacyMultiCoreConfig', 'on')
```

---

**Tip** To display the allowed values of xPC Target environment properties, type `setxpcenv` with no arguments. To display their current values, type `getxpcenv` with no arguments.

---

- If you are doing a network boot and the boot procedure displays a message similar to TFTP Timeout:

- Verify that the `xpctftpserver` program is running. If it is not, recreate the network boot image.
- Temporarily disable the Internet security (firewall) software on the host computer. If you can now boot:
  - Follow the Internet security software instructions to allow the xPC Target boot procedure to work in its presence. For example, add the MATLAB interface to the firewall exception list.
  - Reenable the Internet security software.
- If problems persist, see Chapter 17, “Target Computer Boot Process”.
- If you still cannot boot the target computer from a boot drive, you might need to replace the target computer removable drive, CD disk drive, or network boot image.

### How Do I Fix a Kernel Load Error?

When booting the target computer, you might see a message like the following:

```
xPC Target 4.X loading kernel..@@@@@@@@@@@@@@@@@@@@@@@@
```

The target computer displays this message when it cannot read and load the kernel from the target boot disk.

The probable cause is a bad boot drive. To diagnose this problem, recreate the boot drive. If you have a removable boot drive, reformat the drive or use a new formatted drive and create a new target boot drive. If you have a CD target boot disk, create a new disk. If you are using network boot, recreate the network boot image.

## How Do I Fix a Not Bootable Medium Error?

When booting the target computer, you might get a message similar to the following:

```
Not a bootable medium or NTLDR is missing
```

Selecting either **DOS Loader** or **Standalone** mode instead of **Boot Floppy** mode can cause this message.

To solve this problem:

- 1 If the xPC Target Explorer window is not already open, open it. In the MATLAB Command Window, type `xpcexplr`.

---

**Note** xPC Target Explorer runs only on 32-bit host computers. On 64-bit computers, use the MATLAB command-line interface. For further information, see “Configuring Environment From the MATLAB Command Line” on page 4-21.

---

- 2 In the xPC Target Explorer **xPC Target Hierarchy** pane, select a target computer **Configuration** node. For example, select the **Configuration** node for **TargetPC1**. In the configuration pane, select your desired boot mode.
- 3 Create a new boot disk or network boot image.

### Why Is the Target Computer Halted?

If your target computer displays a System Halted message while booting:

- Verify in xPC Target Explorer that the **TcpIp target driver** parameter is configured as required by your network.
- Recreate the xPC Target boot drive or network boot image and use the new boot medium to boot the target computer.
- Verify that the xPC Target software supports your target computer hardware. Be sure to verify the network communication hardware.



# Modeling

---

- “How Do I Handle Register Rollover for Encoder Blocks?” on page 18-2
- “How Can I Write Custom Device Drivers?” on page 18-3

## How Do I Handle Register Rollover for Encoder Blocks?

Encoder boards have a fixed size counter register of 16 bits, 24 bits, or 32 bits. Regardless of the size, the register always eventually overflows and rolls over. Registers can roll over in either the positive or negative direction.

Some boards provide a hardware mechanism to account for overflows or rollovers. As a best practice, you should design your model to always deal with overflows or rollovers. Defining an initial count can handle the issue for some applications.

To handle register rollovers, you can use standard Simulink blocks to design the following counter algorithm types:

- Rollover Counter — Count the number of times the output of an encoder block has rolled over. It counts up for positive direction rollovers and down for negative direction rollovers.
- Extended Counter — Provides a rollover count not limited by register size. For an  $n$ -bit register, this counter should be able to count values greater than  $2^{(n-1)}$ .

The Incremental Encoder sublibrary of the xPC Target library contains example blocks for these two types of counters. See Rollover Counter and Extended Counter for further details. You can use these blocks in your model as is, or modify them for your model. Connect the output of the encoder block to these blocks.

---

**Note** To view the algorithms used in these implementations, right-click the subsystem and select the **Look Under Mask** option.

---

Keep the following requirements in mind when using these blocks:

- Some driver blocks allow an initial starting value to be loaded into the register. You must pass this value to the rollover blocks to adjust for that offset.
- The rollover block needs to know how many counts each rollover represents. Typically, this number is  $2^n$ , where  $n$  is the size of the register in bits.

## How Can I Write Custom Device Drivers?

You might want to write your own driver if you want to include an unsupported device driver in your xPC Target system. See the *xPC Target Device Drivers Guide* for details.

Before you consider writing custom device drivers for the xPC Target system, you should possess:

- Good C programming skills
- Knowledge of writing S-functions and compiling those functions as C-MEX functions
- Knowledge of SimStruct, a MATLAB Simulink C language header file that defines the Simulink data structure and the SimStruct access macros. It encapsulates all the data relating to the model or S-function, including block parameters and outputs.
- An excellent understanding of the I/O hardware. Because of the real-time nature of the xPC Target system, you must develop drivers with minimal latency. Because most drivers access the I/O hardware at the lowest possible level (register programming), you must have a good understanding of how to control a board with register information and have access to the register-level programming manual for the device.
- A good knowledge of port and memory I/O access over various buses. You need this information to access I/O hardware at the register level.



# Model Compilation

---

- “How Can I Deploy a Standalone Interface to Control a Target Application?” on page 19-2
- “Why Do I Get a Compiler Error When Compiling Models with Links to Dynamic Link Libraries?” on page 19-3
- “Why Do I Get a Compiler Error with Some Blocks and Not Others?” on page 19-4

## **How Can I Deploy a Standalone Interface to Control a Target Application?**

You can use either the xPC Target API dynamic link library (DLL) or the xPC Target component object model (COM) API library to create a custom standalone interface to control a real-time application running on the target computer. To deploy these standalone applications, you must have the xPC Target Embedded Option license. Without this license, you can create and use the standalone application in your environment, but you cannot deploy that application on another host computer that does not contain your licensed copy of the xPC Target software.

See “Embedded Target Boot Method” and *xPC Target API Guide* for details.

## **Why Do I Get a Compiler Error When Compiling Models with Links to Dynamic Link Libraries?**

The xPC Target software supports links to static link libraries (.lib) , not links to dynamic link libraries (.dll). When you compile your models, verify that you link only to static link libraries. Linking to static libraries is not an issue when you compile with xPC Target S-functions.

## **Why Do I Get a Compiler Error with Some Blocks and Not Others?**

Some compilers, such as the WATCOM 1.8 compiler, do not compile all I/O blocks. If a compilation fails, use a Microsoft compiler instead. WATCOM compiler support will be removed in a future release.



# Application Download

---

- “Why Does My Download Time Out?” on page 20-2
- “How Do I Increase the Download Timeout Value?” on page 20-4
- “Why Does the Download Halt Without Completing?” on page 20-5

## Why Does My Download Time Out?

If the host computer and target computer are not connected, or you have not entered the required environment properties, the download process is terminated after about 5 seconds with a timeout error. Be sure that you have followed the instructions outlined in “Installation and Configuration” before continuing.

---

**Note** xPC Target Explorer runs only on 32-bit host computers. On 64-bit computers, use the MATLAB command-line interface. For further information, see “Configuring Environment From the MATLAB Command Line” on page 4-21.

---

To diagnose the problem, use the following procedure:

- 1** If the xPC Target Explorer is not already running, in the MATLAB window, type  
  
`xpcexplr`  
  
The xPC Target Explorer window opens.
- 2** For the target computer in question, check the RS-232 or TCP/IP parameters. Make any required changes to the communication properties and recreate the target boot disk.

---

**Note** RS-232 host-target communication mode will be removed in a future release. Use TCP/IP instead.

---

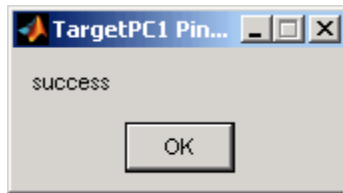
In some cases, the download might have completed even though you get a timeout error. To detect this condition, wait until the target screen displays

`System:initializing application finished.`

- 3** In the xPC Target Explorer window:
  - a** Select the target computer in question.

- b** From the menu bar, select **Target > Ping Target**.

For a working connection between the host computer and target computer, xPC Target Explorer displays a dialog box.



- 4** Right-click the target computer in question and select **Connect**.

If the connection resumes, the connection is all right. If the connection times out consistently for a particular model, the timeout needs to be increased. See “How Do I Increase the Download Timeout Value?” on page 20-4.

For information on setting up the xPC Target software environment, see either “Environment Properties for Serial Communication” or “Environment Properties for Network Communication”, and then see “Target Boot Method”.

## How Do I Increase the Download Timeout Value?

By default, if the host computer does not get a response from the target computer after downloading a target application and waiting about 5 seconds, the host computer software times out. On the other hand, the target computer responds only after downloading and initializing the target application.

Usually 5 seconds is enough time to download a target application, but in some cases it may not be long enough. The time to download a target application mostly depends on your I/O hardware. For example, thermocouple hardware takes longer to initialize. In this case, even though the target computer is fine, a false timeout is reported.

You can increase the timeout value in one of the following ways:

- At the model level, open the **Simulink > Configuration Parameters** dialog box and navigate to the **xPC Target options** node. Clear the **Use default communication timeout** parameter and enter a new desired timeout value in the **Specify the communication timeout in seconds** parameter. For example, enter 20 to increase the value to 20 s.
- At the target application level, use the target application `xpc.target.xpc.set (target application object)` method to set the `CommunicationTimeOut` property to the desired timeout value. For example, to increase the value to 20 s:

```
tg.set('CommunicationTimeOut',20)
```

For both methods, the host computer polls the target computer about once every second, and if a response is returned, returns the success value. Only if a download really fails does the host computer wait the full twenty seconds.

## Why Does the Download Halt Without Completing?

If the MATLAB interface freezes and there are target ping errors, this failure is likely the result of an active firewall, a long initialization process, or both combined. To diagnose this problem, see:

- “Is There Communication Between Your Computers?” on page 16-2
  - “Are I/O Boards with Slow Initialization Times Causing Communication Problems During Download?” on page 16-4
- “Are Multiple Ethernet Cards in the Target Computer Causing Communication Problems During Download?” on page 16-6



# Application Execution

---

- “How Can I View the Contents of the Target Computer Display on the Host Computer?” on page 21-2
- “Why Is My Requested Sample Time Different from the Measured Sample Time?” on page 21-3
- “What Sample Time Can I Expect from a Target Application?” on page 21-5
- “Why Has the Stop Time Changed?” on page 21-6

## **How Can I View the Contents of the Target Computer Display on the Host Computer?**

From the host computer, you can view the target computer monitor with the MATLAB `xpctargetspy` command. For example, type:

```
xpctargetspy('TargetPC1')
```

The Real-Time xPC Target Spy window is displayed on the host computer monitor.



## Why Is My Requested Sample Time Different from the Measured Sample Time?

You might notice that the sample time you request does not equal the actual sample time you measure from your model. This difference depends on your hardware. Your model sample time is as close to your requested time as the hardware allows.

However, hardware does not allow infinite precision in setting the spacing between the timer interrupts. This limitation can cause the divergent sample times.

For all PCs, the only timer that can generate interrupts is based on a 1.193 MHz clock. For the xPC Target system, the timer is set to a fixed number of ticks of this frequency between interrupts. If you request a sample time of 1/10000, or 100, microseconds, you do not get exactly 100 ticks. Instead, the xPC Target software calculates that number as

$$100 \times 10^{-6} \text{ seconds} \times 1.193 \times 10^6 \text{ ticks/seconds} = 119.3 \text{ ticks}$$

The xPC Target software rounds this number to the nearest whole number, 119 ticks. The actual sample time is then

$$119 \text{ ticks} / (1.193 \times 10^6 \text{ ticks/second}) = 99.75 \times 10^{-6} \text{ seconds} \\ (99.75 \text{ microseconds})$$

Compared to the requested original sample time of 100 microseconds, this value is 0.25% faster.

As an example of how you can use this value to derive the expected deviation for your hardware, assume the following:

- Output board that generates a 50 Hz sine wave (expected signal)
- Sample time of 1/10000
- Measured signal of 50.145 Hz

The difference between the expected and measured signals is .145, which deviates from the expected signal value by 0.29% (0.145/50). Compared to the

previously calculated value of 0.25%, there is a difference of 0.04% from the expected value.

If you want to further refine the measured deviation for your hardware, assume the following:

- Output board that generates a 50 Hz sine wave (expected signal)
- Sample time of 1/10200
- Measured signal of 50.002 Hz

$$1/10200 \text{ seconds} \times 1.193 \times 10^6 \text{ ticks/seconds} = 116.96 \text{ ticks}$$

Round this number to the nearest whole number of 117 ticks. The resulting frequency is then

$$(116.96 \text{ ticks}/117)(50) = 49.983 \text{ Hz}$$

The difference between the expected and measured signal is 0.019, which deviates from the expected signal value by 0.038% (0.019/50.002). The deviation when the sample time is 1/10000 is 0.04%.

Some amount of error is common for most PCs, and the margin of error varies from machine to machine.

---

**Note** Most high-level operating systems, like Microsoft Windows or Linux<sup>®</sup>, occasionally insert extra long intervals to compensate for errors in the timer. Be aware that the xPC Target software does not attempt to compensate for timer errors. For this product, close repeatability is more important for most models than exact timing. However, some chips might have inherent designs that produce residual jitters that could affect your system. For example, some Pentium chips might produce residual jitters on the order of 0.5 microsecond from interrupt to interrupt.

---

## What Sample Time Can I Expect from a Target Application?

The xPC Target kernel is tuned for minimal overhead and maximum performance. On the target computer, the kernel dedicates all of its resources to the target application. To check what sample time you can expect, run `xpcbench` at the MATLAB command line.

- `xpcbench('this')` — Evaluates your target computer against predefined benchmarks and compares it to other target computers. The results indicate the smallest base sample time that an xPC Target application can achieve on your system.
- `xpcbench('model.mdl')` — Evaluates your target computer against your specific model.

Actual obtainable sample times depend on a number of factors, including:

- Processor performance
- Model complexity
- I/O block types
- Number of I/O channels

## Why Has the Stop Time Changed?

If you change the step size of a target application after it has been built, it is possible that the target application will execute for fewer steps than you expect. The number of execution steps is:

```
floor(stop time/step size)
```

When you compile code for a model, Simulink Coder calculates a number of steps based on the current step size and stop time. If the stop time is not an integral multiple of the step size, Simulink Coder adjusts the stop time for that model based on the original stop time and step size. If you later change a step size for a target application but do not recompile the code, xPC Target uses the new step size and the previously adjusted stop time. The resulting model may execute for fewer steps than you expect.

For example, if a model has a stop time of 2.4 and a step size of 1, Simulink Coder adjusts the stop time of the model to 2 at compilation. If you change the step size to 0.6 but do not recompile the code, the expected number of steps is 4, but the actual number of steps is 3 because xPC Target uses the previously adjusted stop time of 2.

To avoid this problem, verify that the original stop time (as specified in the model) is an integral multiple of the original step size.

# Application Parameters

---

- “Why Does the getparamid Function Return Nothing?” on page 22-2
- “Can I Tune All the Parameters in the Model?” on page 22-3

## Why Does the `getparamid` Function Return Nothing?

The `xpctarget.xpc.getparamid` and `xpctarget.xpc.getsignalid` functions accept `block_name` parameters. For these functions, enter for `block_name` the mangled name that the Simulink Coder software uses for code generation. You can determine the `block_name` as follows:

- If you do not have special characters in your model, use the `gcb` function.
- If the blocks of interest have special characters, retrieve the mangled name with `tg.showsignals='on'` or `tg.showparam = 'on'`.

For example, if carriage return `'\n'` is part of the block path, the mangled name returns with carriage returns replaced by spaces.

## **Can I Tune All the Parameters in the Model?**

Observable parameters are those you can tune. Nonobservable parameters are those that exist in the target application, but are not tunable from the host computer. You cannot tune parameters of complex or multiword data types. You can tune fixed-point data types, including boolean, integer, and double. For more on fixed-point data types, see “Data Type Support”.





# Application Signals

---

- “Why Do I Get Error -10: Invalid File ID on the Target Computer?” on page 23-2
- “Can I Access All the Signals in the Model?” on page 23-3

## **Why Do I Get Error -10: Invalid File ID on the Target Computer?**

You might get this error if you are acquiring signal data with a file scope. This error occurs because the size of the signal data file exceeds the available space on the disk. The signal data is most likely corrupt and irretrievable. You should delete the signal data file and reboot the xPC Target system. To prevent this occurrence, monitor the size of the signal data file as the scope acquires data.

Refer to the MathWorks Support xPC Target Web site (<http://www.mathworks.com/support/product/XP>) for additional information.

## Can I Access All the Signals in the Model?

You cannot directly access or tag signals from virtual buses or blocks. To observe a virtual block:

- 1** Add a unity Gain block (a Gain block with a gain of 1.0) to the model.
- 2** Connect the signal output of the virtual block to the input of the unity Gain block.
- 3** Access or tag the output signal of the unity Gain block.

To observe a virtual bus, add a unity Gain block to each individual signal.

You cannot directly access signals you have optimized with block reduction optimization. Access these signals by making them test points.

You cannot access signals of complex or multiword data types.



# Application Performance

---

- “How Can I Improve Runtime Performance?” on page 24-2
- “Why Does Running My Model Cause CPU Overload Messages on the Target Computer?” on page 24-4
- “How Small a Sample Time Can I Use Without CPU Overload?” on page 24-6
- “Can I Allow CPU Overloads?” on page 24-7

## How Can I Improve Runtime Performance?

To improve runtime performance and reduce the task execution time (TET) of a model `model.mdl`:

- 1 Run `xpcbench` at the MATLAB command line:
  - `xpcbench('this')` — Evaluates your target computer against predefined benchmarks and compares it to other target computers. The results indicate the smallest base sample time that an xPC Target application can achieve on your system.
  - `xpcbench('model.mdl')` — Evaluates your target computer against your specific model.

---

**Tip** For more information on xPC Target benchmarking, see <http://www.mathworks.com/support/product/XP/-productnews/benchmarks.html>.

---

- 2 If your target computer is not high on the list of benchmark computers, consider switching to a target computer with higher performance.
- 3 Run the xPC Target profiler on `model.mdl` and record where the time is being spent. (See “Profiling Target Application Execution” on page 26-6.)
- 4 Perform optimizations such as the following:
  - Disable TET logging for the application. To do this, clear the **Log Task Execution Time** check box in the **xPC Target options** pane of the Configuration Parameters dialog box.
  - Disable log output for the application. To do this, clear the following check boxes:
    - **Time**
    - **States**
    - **Output**
    - **Final states**

---

- **Signal logging**

in the **Data Import/Export** pane of the Configuration Parameters dialog box.

- Use polling mode, if you do not need background processes (see “Polling Mode” on page 6-5 for more on setting this mode).
  - Increase **Fixed-step size (fundamental sample time)** in the **Solver** pane of the Configuration Parameters dialog box.
  - Disable the target scope display. To do this, clear the **Enable target scope** check box in the **Appearance** pane of xPC Target Explorer.
  - Use fewer scopes in the model.
  - Reduce the number of I/O channels in the model.
- 5** Consider partitioning the model and running it on a multicore system. (See “Configuring Models for Targets with Multicore Processors”.)

---

**Note** You must set the **Multicore CPU support** check box in the **Settings** pane of xPC Target Explorer to allow xPC Target to use your target computer in multicore mode.

---

- 6** Consider partitioning the model and running it on multiple target computers. This optimization might require multitarget synchronization using CAN, UDP, parallel port, or reflective memory.
- 7** Check the other questions under Chapter 24, “Application Performance” for tips on eliminating CPU overloads and improving task execution time.
- 8** Check the MathWorks Support Web site and MATLAB Central for other tips. See “How Do I Find the MathWorks Support Web Site?” on page 25-2.
- 9** Call MathWorks Technical Support. See “How Do I Contact MathWorks Technical Support?” on page 25-5.

## Why Does Running My Model Cause CPU Overload Messages on the Target Computer?

A CPU overload indicates that the CPU was unable to complete processing a model time step before being asked to restart. When an overload occurs, one of the following can happen:

- The xPC Target kernel halts model execution.
- If the overload is allowed, the model execution continues until a predefined event (see “Can I Allow CPU Overloads?” on page 24-7 for details). If a model continues running after a CPU overload, the model time step is as long as the time required to finish the execution. This behavior delays the following time step.

This error might occur if you have:

- **Real CPU overloads** — Those caused by model design or target computer resources. For example, a model is trying to do more than can be done in the allocated time on the target computer. Possible reasons are:
  - The target computer is too slow or the model sample time is too small (see “How Small a Sample Time Can I Use Without CPU Overload?” on page 24-6).
  - The model is too complex (algorithmic complexity).
  - I/O latency, where each I/O channel used introduces latency into the system. This might cause the execution time to exceed the model time step.

To address I/O latency, you can use the xPC Target Interactive Guide ([http://www.mathworks.com/support/product/XP/productnews/-interactive\\_guide/xPC\\_Target\\_Interactive\\_Guide.html](http://www.mathworks.com/support/product/XP/productnews/-interactive_guide/xPC_Target_Interactive_Guide.html)) to find latency numbers for boards supported by the block library. For example, if your application includes the National Instruments® PCI-6713 board, and you want to use four outputs:

- 1 Look up the board in the xPC Target Interactive Guide.

From the table, the D/A latency is 1+2.4N.

- 2 To get the latency for four outputs, calculate the latency



$$1+(2.4 \times 4) = 10.6 \text{ microseconds}$$

**3** Include this value in your sample time calculations.

- **Spurious CPU overloads** — Commonly caused by factors outside of the model design. These overloads are most likely caused by one of the following:
  - Advanced Power Management
  - Plug-and-Play (PnP) operating system
  - System Management Interrupts (SMIs)

Enabling any of these properties causes non-real-time behavior from the target computer. You must disable these BIOS properties for the target computer to run the target application in real time. See “Target Computer BIOS Settings”.

---

**Note** You cannot always disable SMIs from your BIOS. However, for some chipsets, you can programmatically prevent or disable SMIs. For example, see the *Disabling SMIs on Intel ICH5 Chipsets* document at MATLAB Central for a solution to disabling SMIs in the Intel® ICH5 family.

---

For further information and test models, see the *xPC Target CPU Overloads* document at MATLAB Central.

## How Small a Sample Time Can I Use Without CPU Overload?

If the model has too small a sample time, a CPU overload can occur. This error indicates that to run the target application, executing one step of the model requires more time than the sample time for the model (`Fixed step size` property) allows.

When this error occurs, the target object property `CPUOverload` changes from `none` to `detected`. To diagnose the issue:

**1** Run `xpcbench` at the MATLAB command line:

- `xpcbench('this')` — Evaluates your target computer against predefined benchmarks and compares it to other target computers. The results indicate the smallest base sample time that an xPC Target application can achieve on your system.
- `xpcbench('model.mdl')` — Evaluates your target computer against your specific model.

---

**Tip** For more information on xPC Target benchmarking, see <http://www.mathworks.com/support/product/XP/-productnews/benchmarks.html>.

---

**2** Change the model `Fixed step size` property to at least the indicated value and rebuild the model. Use the **Solver** node in the Simulink model Configuration Parameters dialog.

**3** If these steps do not solve your problem, see:

“How Can I Improve Runtime Performance?” on page 24-2.

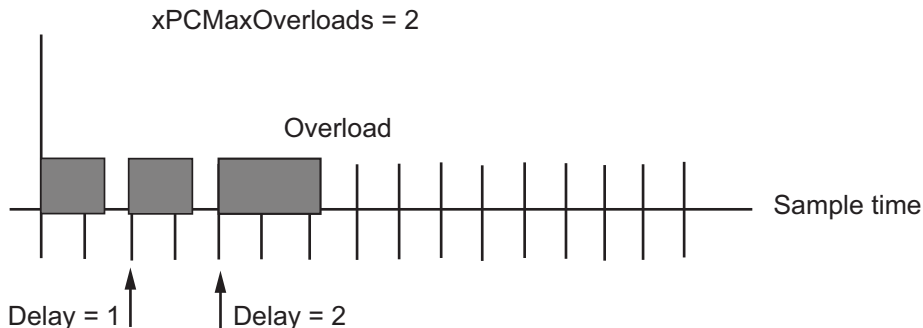
## Can I Allow CPU Overloads?

Typically, the xPC Target kernel halts model execution when it encounters a CPU overload. You can direct the xPC Target environment to allow CPU overloads using the following options in the **TLC options** parameter in the Code Generation pane of the Simulink Configuration Parameters dialog box.

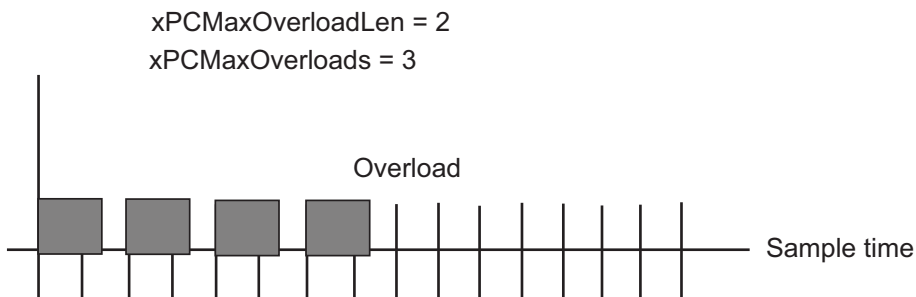
Option	Description	Default
xPCMaxOverloads	Number of acceptable overloads.	0
xPCMaxOverloadLen	Number of contiguous acceptable overloads. If you do not specify this option, the default value is the same as xPCMaxOverloads. Specify a value that is the same or less than the value for the xPCMaxOverloads option. You should not use a value greater than xPCMaxOverloads.	Same as value of xPCMaxOverloads
xPCStartupFlag	Number of executions of the model at startup, where the timer interrupt is temporarily disabled during model execution. After the model finishes the first xPCStartupFlag number of executions, the xPC Target software enables the timer interrupt, which will invoke the next execution for the model.	1

If you experience a CPU overload after the model starts, the software ignores timer interrupts if the task is already running. The model continues running, subject to the values of xPCMaxOverloads and xPCMaxOverloadLen. The model then executes at the next step.

The following graphic illustrates a sample time line for what happens if the `xPCMaxOverloads` property is 2. In this case, the first overload is less than 2, so the model executes at the next time step, assuming a time delay of 1 step.

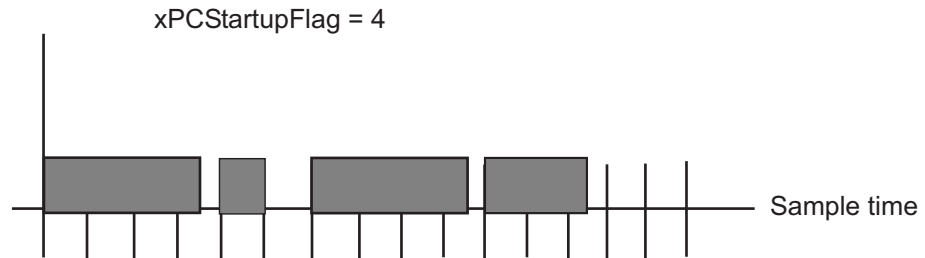


The following graphic illustrates a sample time line for what happens if the `xPCMaxOverloads` property is 3 and `xPCMaxOverloadLen` is 2. In this case, the first 3 overloads are allowed, the fourth overload causes the model to stop.



If the overload occurs at model start, the model continues running after a CPU overload. The model time step is as long as the time required to finish the execution. This behavior delays the following time step.

The following graphic illustrates a sample time line for what happens if the `xPCStartupFlag` property is 4. In this case, the kernel ignores all overloads for the first four steps. All overloads in this graphic are within 4 steps.



The following example describes the interaction of the three properties. When the xPC Target kernel runs the model, it checks the number of CPU overloads against the values of `xPCMaxOverloads` and `xPCMaxOverloadLen`. When the number of CPU overloads reaches the lower of these two values, the xPC Target kernel stops executing the model.

For example, if you enter a line like the following for the **TLC options** parameter:

```
-axPCMaxOverloads=30 -axPCOverLoadLen=2 -axPCStartupFlag=5
```

the xPC Target software ignores CPU overloads for the first five iterations through the model. After this, the xPC Target software allows up to 30 CPU overloads, preventing no more than two consecutive CPU overloads.

With the TLC options, you can use the following blocks in your model to help keep track of the number of CPU overloads.

- Use the xPC Target Get Overload Counter and xPC Target Set Overload Counter blocks to set and keep track of CPU overload numbers.
- Use the Pentium Time Stamp Counter block to profile your model.



# Getting MathWorks Support

---

- “How Do I Find the MathWorks Support Web Site?” on page 25-2
- “How Do I Get an Updated Software Release?” on page 25-3
- “What Should I Do After I Get a New Release?” on page 25-4
- “How Do I Contact MathWorks Technical Support?” on page 25-5

## How Do I Find the MathWorks Support Web Site?

For xPC Target solutions and guidelines, see the following MathWorks Web site resources:

- MathWorks Support (<http://www.mathworks.com/support/product/XP>)

The xPC Target documentation is also available from this site.

- MATLAB Central File Exchange for xPC Target Product (<http://www.mathworks.com/matlabcentral/fileexchange/?term=productid:118>)



## How Do I Get an Updated Software Release?

- 1** In the MATLAB Command Window, select **Start > Simulink > xPC Target > Product Page (Web)** .
- 2** Look for the section on downloading software and download the version you want.
- 3** Install and integrate the new release software.
- 4** Recreate your xPC Target environment. See “What Should I Do After I Get a New Release?” on page 25-4

## What Should I Do After I Get a New Release?

If you are working with a new xPC Target release, either downloaded from the MathWorks Web site ([http://www.mathworks.com/web\\_downloads/](http://www.mathworks.com/web_downloads/)) or installed from a DVD, you must do the following:

- 1 In the MATLAB Command Window, type:

```
xpcexplr
```

---

**Note** xPC Target Explorer runs only on 32-bit host computers. On 64-bit computers, use the MATLAB command-line interface. For further information, see “Configuring Environment From the MATLAB Command Line” on page 4-21.

---

- 2 Recreate your xPC Target environment (see “Serial Communication” or Network Communication in the *xPC Target Getting Started Guide*).
- 3 Create a new boot drive or image.
- 4 Reboot the target computer.
- 5 Rebuild target applications for that new xPC Target release.

## How Do I Contact MathWorks Technical Support?

- 1 If you cannot solve your problem, call function `getxpcinfo` to retrieve diagnostic information for your xPC Target configuration. This function writes the diagnostic information to the file `xpcinfo.txt` in the current folder.

---

**Note** The `xpcinfo.txt` file might contain information sensitive to your organization. Review the contents of this file before disclosing it to MathWorks.

---

- 2 Contact MathWorks directly for online or phone support:  
[http://www.mathworks.com/support/contact\\_us](http://www.mathworks.com/support/contact_us)



# Tuning Performance

---

- “Building Referenced Models in Parallel” on page 26-2
- “Multicore Processor Configuration” on page 26-4
- “Profiling Target Application Execution” on page 26-6

## Building Referenced Models in Parallel

The xPC Target software allows you to build referenced models in parallel on a compute cluster. In this way, you can more quickly build and download xPC Target applications to the target computer.

---

**Note** The following procedure assumes you have a functioning xPC Target installation on your host computer.

---

- 1 Identify a set of worker computers, which might be separate cores on your host computer or computers in a remote cluster running under Windows.
- 2 If you intend to use separate cores on the host computer, install Parallel Computing Toolbox™ on the host computer.
- 3 If you intend to use computers in a remote cluster:
  - a Install the following on each cluster computer:
    - MATLAB
    - Parallel Computing Toolbox
    - MATLAB Distributed Computing Server™
    - xPC Target
    - Build compiler

---

**Tip** Install the same compiler and compiler version at the same location as on the host computer.

---

- b Start and configure the remote cluster according to the instructions in *MATLAB Distributed Computing Server System Administrator's Guide*.
- 4 Run MATLAB on the host computer.
- 5 In MATLAB, type `matlabpool` to open a MATLAB pool.

- 6 Type `pctRunOnAll` to configure the compiler for all of the remote workers.  
For example:

```
pctRunOnAll('xpcsetCC(''VisualC'',  
            ''C:\Program Files\Microsoft Visual Studio 9.0''))
```

In this configuration, the host computer and all of the remote workers have installed Microsoft Visual Studio 9.0 at `C:\Program Files\Microsoft Visual Studio 9.0`.

- 7 Build and download your model.

See “Reduce Build Time for Referenced Models” in the *Simulink Coder User’s Guide* for more about increasing the speed of parallel builds.

## Multicore Processor Configuration

For better performance on your target computer, you can run multirate target applications on multiple cores. Use this capability if your target computer has a multicore processor and you want to take advantage of it for multirate models. Before you consider enabling this capability, see “Target Computer BIOS Settings” for the effects of BIOS settings.

To build and download multirate models on your multiple core target computer:

- 1 In the **Settings** node of xPC Target Explorer, select the **Multicore CPU support** check box.
- 2 Open your model in the Simulink model editor.
- 3 Add a Rate Transition block to transition between rates.

---

**Note** Multirate models must use Rate Transition blocks. If your model uses other blocks for rate transitions, building the model generates an error.

---

- 4 Select the **Ensure data integrity during data transfer** check box of the Rate Transition block.
- 5 Clear the **Ensure deterministic data transfer (maximum delay)** check box of the Rate Transition block. This forces the Rate Transition block to use the most recent data available.

---

**Note** Because this box is cleared, the transferred data might differ from run to run.

---

- 6 In the Simulink model editor, select **View > Model Explorer**.
- 7 In Simulink Model Explorer, right click in the **Model Hierarchy** pane and select **Configuration > Add configuration for concurrent execution**



- 8** In the new configuration, select **Solver**.
- 9** Check **Enable concurrent tasking**.
- 10** Click **Configure Tasks**.

For more on configuring your model for concurrent execution, see “Configuring Models for Targets with Multicore Processors”.

## Profiling Target Application Execution

### In this section...

“Profiling Overview” on page 26-6

“Configuring Your Model to Collect Profile Data During Execution” on page 26-6

“Displaying and Evaluating Profile Data” on page 26-7

### Profiling Overview

You can profile your target computer to see the execution sequence of your target application and then tune the performance. This process is especially useful if your target application is configured to take advantage of multicore processors on the target computer.

To configure your target computer and model to take advantage of multicore processors, see “Multicore Processor Configuration” on page 26-4.

Profiling your target computer requires these steps:

- 1 Configure the model to enable the collection of profile data during execution.
- 2 Display and evaluate the profile data.

Profiling adds a slight increase to the execution time of the target application.

### Configuring Your Model to Collect Profile Data During Execution

To configure your model to collect profile data during execution:

- 1 Open your model in the Simulink model editor.
- 2 Select the **Simulation > Configuration Parameters > Code Generation > xPC Target options node > Enable profiling**.
- 3 In the same **Configuration Parameters** pane, consider the value of the **Number of events** parameter.

By default, the software logs 5000 events for profiling. You can increase or decrease this number. When the software logs the specified number of events or the model stops, the software stops collecting the data and writes it to *current\_working\_folder*\xPCTrace.csv on the target computer.

**4** Save your changes.

To try this procedure with a preconfigured model, use:

```
matlabroot\toolbox\rtw\targets\xpc\xpcdemos\xpcprofdemo.mdl
```

To build, download, and display the profile data, use a profiling script as described in “Displaying and Evaluating Profile Data” on page 26-7.

## Displaying and Evaluating Profile Data

To see the profile data that your model collects during execution on the target computer, you run a profiling script.

If you have not yet done so, see “Configuring Your Model to Collect Profile Data During Execution” on page 26-6.

In the MATLAB Command Window, run the script:

```
matlabroot\toolbox\rtw\targets\xpc\xpcdemos\profile_xpc_demo.m
```

This script:

- 1** Builds and downloads the model that you used to collect the profile data. By default, the script uses *matlabroot*\toolbox\rtw\targets\xpc\xpcdemos\xpcprofdemo.mdl.
- 2** Saves the data in xPCTrace.csv on the target computer.

xPCTrace.csv is a raw data file that contains information such as a header, version number, row in which data starts, CPU frequency, and the time of the first event.

- 3** Transfers xPCTrace.csv from the target computer to the current working folder of the target computer.

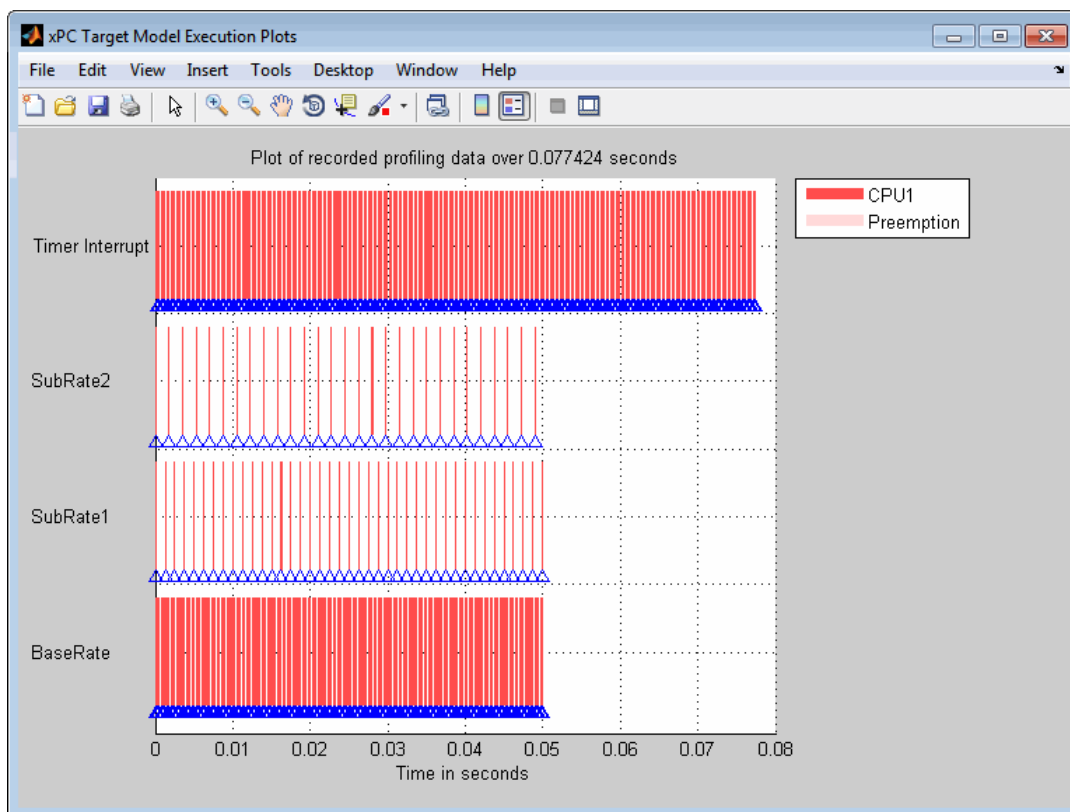
- 4 On the host computer, organizes the raw data into the `profileInfo` structure and displays the profile data in a MATLAB figure window and an HTML file.

To try this procedure with a demo model, use `profile_xpc_demo.m`. This demo calls the script `matlabroot\toolbox\rtw\targets\xpc\xpcdemos\profile_xpc_demo.m`.

To evaluate the displays, see “Interpreting Profile Data” on page 26-8.

### Interpreting Profile Data

The MATLAB figure window displays a task activity graph with the following entries:



<b>Row</b>	<b>Description</b>
Timer interrupt	Recorded polling data for interrupts
SubRate2	Recorded polling data for SubRate2 task
SubRate1	Recorded polling data for SubRate1 task
BaseRate	Recorded polling data for BaseRate task

The HTML report displays model execution profile results for each task:

xPC Target Report - Execution Profile Results

File Edit View Go Debug Desktop Window Help

Location: /Users/Guest/AppData/Local/Temp/xpc\_profile\_2010\_11\_2\_12\_28\_31.html

## Model Execution Profiling Results

- [Analysis of recorded profiling data](#)

All times are in seconds. The CPU frequency is 2.7932e+009 Hz, and the timer resolution is 3.5801e-010 seconds.

---

### Analysis of profiling data recorded over 0.077424 seconds.

Profiling data was recorded over 0.077424 seconds. The recorded data for [task turnaround times](#) and [task execution times](#) is presented in the table below.

Task	Maximum turnaround time	Average turnaround time	Maximum execution time	Average execution time	Average sample time
BaseRate	2.72e-005 at 6.5e-006	1.58e-005	2.72e-005 at 6.5e-006	1.58e-005	0.00024973
SubRate1	3.42e-006 at 3.45e-005	1.15e-006	3.42e-006 at 3.45e-005	1.15e-006	0.0012484
SubRate2	1.7e-006 at 3.87e-005	1.15e-006	1.7e-006 at 3.87e-005	1.15e-006	0.0017475
Timer Interrupt	2.23e-006 at 0.000499	1.64e-006	2.23e-006 at 0.000499	1.64e-006	0.00024975

If you have any values shown as N/A (Not Available), this may be due not enough data collected.

**Task turnaround time** is the elapsed time between start and finish of the task. If the task is not preempted, the task turnaround time equals the task execution time.

**Task execution time** is the time between task start and finish, when the task is actually running and not preempted by another task. The task execution time cannot be measured directly; it is inferred from the task start and finish time and the intervening periods during which another task has preempted it. Note that, in performing these calculations, processor time consumed by the scheduler while switching tasks is not taken into account. This means that, in cases where preemption occurs, the reported task execution times overestimate the true values.

<b>Result</b>	<b>Description</b>
Maximum turnaround time	Longest time between when the task starts and finishes. This time includes task preemptions (interrupts). If no preemptions occur, the maximum turnaround time is the same as the maximum task execution time.
Average turnaround time	Average time between when the task starts and finishes. This time includes task preemptions (interrupts). If no preemptions occur, the average turnaround time is the same as the average task execution time.
Maximum execution time	Longest time between when the task starts and finishes. This time does not include task preemptions (interrupts).
Average execution time	Average time between when the task starts and finishes. This time does not include task preemptions (interrupts).
Average sample time	Average sample time of the task.

### **Customizing profile\_xpc\_demo.m**

The profiling script, `profile_xpc_demo.m`, works with the demo model:

```
matlabroot\toolbox\rtw\targets\xpc\xpcdemos\xpcprofdemo.mdl
```

To customize this script:

- 1** Configure your model as described in “Configuring Your Model to Collect Profile Data During Execution” on page 26-6.
- 2** Copy  
`matlabroot\toolbox\rtw\targets\xpc\xpcdemos\profile_xpc_demo.m`  
to your working area and rename the copy. For example, use  
the name `my_profile_xpc_demo.m`.
- 3** Edit `my_profile_xpc_demo.m`.

- 4 Notice that the `my_profile_xpc_demo.m` script uses the `profileInfo` structure, which allows you to customize the script. This structure contains the following fields:

Field	Description	Values
<code>rawdtaonhost</code>	Location of the raw data. If this field does not exist, the script assumes a value of 0.	0 — Default. Transfer the raw data from the target computer to the host.  1 — Look for the saved raw data file <code>xPCTrace.csv</code> in the current folder on the host computer.
<code>modelName</code>	Name of the model	<code>xpcprofdemo.mdl</code> — Default. The name of a model in the <code>xpcdemos</code> folder.  <code>your_model_name.mdl</code> — Specify the name of your model that you want to profile.
<code>noplot</code>	Specify whether or not to display the model execution plots on the host computer monitor.	0 — Default. Do not display the model execution plots on the host computer monitor.  1 — Display the model execution plots on the host computer monitor.
<code>noreport</code>	Specify whether or not to display the model profiling result report on the host computer monitor.	0 — Default. Do not display the model profiling result report on the host computer monitor.  1 — Display the model profiling result report on the host computer monitor.



- 5** Change values as desired. For example, to profile your own model, replace instances of `xpcprofdemo.mdl` with your model name, such as `my_xpcprofdemo.mdl`.
- 6** Run your custom script.



# Function Reference

---

Classes (p. 27-2)	xPC Target .NET class descriptions
Target Computers (p. 27-3)	Control target computer hardware and operating system
Target Environments (p. 27-4)	Manage target computer environment collection objects
Target Applications (p. 27-5)	Control target application on target computer
Scopes (p. 27-6)	Control scopes on target computer
Parameters (p. 27-7)	Read and update target application parameters
Signals (p. 27-8)	Read and update signal values
Data Logs (p. 27-9)	Log and read back target computer data
File Systems (p. 27-10)	Control target computer file system and FTP communication with target computer

## Classes

<code>xpctarget</code> Package	Package for all xPC Target MATLAB classes
<code>xpctarget.env</code> Class	Stores target environment properties
<code>xpctarget.fs</code> Class	Manage the directories and files on the target computer
<code>xpctarget.fsbase</code> Class	Base class of file system and file transfer protocol (FTP) classes
<code>xpctarget.ftp</code> Class	Manage the directories and files on the target computer via file transfer protocol (FTP)
<code>xpctarget.targets</code> Class	Container object to manage target computer environment collection objects
<code>xpctarget.xpc</code> Class	Target object representing target application
<code>xpctarget.xpcfsc</code> Class	Control and access properties of file scopes
<code>xpctarget.xpcsc</code> Class	Base class for all scope classes
<code>xpctarget.xpcscho</code> Class	Control and access properties of host scopes
<code>xpctarget.xpcsc</code> Class	Control and access properties of target scopes

## Target Computers

<code>getxpcinfo</code>	Retrieve diagnostic information to help troubleshoot configuration issues
<code>macaddr</code>	Convert string-based MAC address to vector-based one
<code>xpcbench</code>	xPC Target benchmark
<code>xpcbootdisk</code>	Create xPC Target boot disk or DOS Loader files and confirm current environment properties
<code>xpcbytes2file</code>	Generate file suitable for use by From File block
<code>xpcexplr</code>	Open xPC Target Explorer
<code>xpcgetCC</code>	Compiler settings for xPC Target environment
<code>xpcnetboot</code>	Create kernel to boot target computer over dedicated network
<code>xpcsetCC</code>	Compiler settings for xPC Target environment
<code>xpctarget.xpc</code>	Create target object representing target application
<code>xpctarget.xpc.getxpcpci</code>	Determine which PCI boards are installed in target computer
<code>xpctarget.xpc.targetping</code>	Test communication between host and target computers
<code>xpctargetping</code>	Test communication between host and target computers
<code>xpctargetspy</code>	Open Real-Time xPC Target Spy window on host computer
<code>xpctest</code>	Test xPC Target installation
<code>xpcwwenable</code>	Disconnect target computer from current client application

## Target Environments

<code>getxpcenv</code>	List environment properties assigned to MATLAB variable
<code>setxpcenv</code>	Change xPC Target environment properties
<code>xpctarget.env.get (env object)</code>	Return target environment property values
<code>xpctarget.env.set (env object)</code>	Change target environment object property values
<code>xpctarget.targets</code>	Create container object to manage target computer environment collection objects
<code>xpctarget.targets.Add (env collection object)</code>	Add new xPC Target environment collection object
<code>xpctarget.targets.get (env collection object)</code>	Return target object collection environment property values
<code>xpctarget.targets.getTargetNames (env collection object)</code>	Retrieve xPC Target environment object names
<code>xpctarget.targets.Item (env collection object)</code>	Retrieve specific xPC Target environment (env) object
<code>xpctarget.targets.makeDefault (env collection object)</code>	Set specific target computer environment object as default
<code>xpctarget.targets.Remove (env collection object)</code>	Remove specific xPC Target environment object
<code>xpctarget.targets.set (env collection object)</code>	Change target object environment collection object property values

## Target Applications

<code>xpctarget.xpc</code>	Create target object representing target application
<code>xpctarget.xpc.close</code>	Close serial port connecting host computer with target computer
<code>xpctarget.xpc.get (target application object)</code>	Return target application object property values
<code>xpctarget.xpc.load</code>	Download target application to target computer
<code>xpctarget.xpc.reboot</code>	Reboot target computer
<code>xpctarget.xpc.set (target application object)</code>	Change target application object property values
<code>xpctarget.xpc.start (target application object)</code>	Start execution of target application on target computer
<code>xpctarget.xpc.stop (target application object)</code>	Stop execution of target application on target computer
<code>xpctarget.xpc.unload</code>	Remove current target application from target computer

## Scopes

<code>xpctarget.xpc</code>	Create target object representing target application
<code>xpctarget.xpc.addscope</code>	Create scopes
<code>xpctarget.xpc.getscope</code>	Scope object pointing to scope defined in kernel
<code>xpctarget.xpc.remscope</code>	Remove scope from target computer
<code>xpctarget.xpcsc.addsignal</code>	Add signals to scope represented by scope object
<code>xpctarget.xpcsc.get (scope object)</code>	Return property values for scope objects
<code>xpctarget.xpcsc.remsignal</code>	Remove signals from scope represented by scope object
<code>xpctarget.xpcsc.set (scope object)</code>	Change property values for scope objects
<code>xpctarget.xpcsc.start (scope object)</code>	Start execution of scope on target computer
<code>xpctarget.xpcsc.stop (scope object)</code>	Stop execution of scope on target computer
<code>xpctarget.xpcsc.trigger</code>	Software-trigger start of data acquisition for scope(s)



## Parameters

<code>xpctarget.xpc</code>	Create target object representing target application
<code>xpctarget.xpc.getparam</code>	Value of target object parameter index
<code>xpctarget.xpc.getparamid</code>	Parameter index from parameter list
<code>xpctarget.xpc.getparamname</code>	Block path and parameter name from index list
<code>xpctarget.xpc.loadparamset</code>	Restore parameter values saved in specified file
<code>xpctarget.xpc.saveparamset</code>	Save current target application parameter values
<code>xpctarget.xpc.setparam</code>	Change writable target object parameters

## Signals

<code>xpctarget.xpc</code>	Create target object representing target application
<code>xpctarget.xpc.getsignal</code>	Value of target object signal index
<code>xpctarget.xpc.getsignalid</code>	Signal index or signal property from signal list
<code>xpctarget.xpc.getsignalidsfromlabel</code>	Return vector of signal indices
<code>xpctarget.xpc.getsignallabel</code>	Return signal label
<code>xpctarget.xpc.getsignalname</code>	Signal name from index list

## Data Logs

`xpctarget.xpc`

Create target object representing target application

`xpctarget.xpc.getlog`

All or part of output logs from target object

## File Systems

<code>xpctarget.fs</code>	Create xPC Target file system object
<code>xpctarget.fs.diskinfo</code>	Information about target computer drive
<code>xpctarget.fs.fclose</code>	Close open target computer file(s)
<code>xpctarget.fs.fileinfo</code>	Target computer file information
<code>xpctarget.fs.filetable</code>	Information about open files in target computer file system
<code>xpctarget.fs.fopen</code>	Open target computer file for reading
<code>xpctarget.fs.fread</code>	Read open target computer file
<code>xpctarget.fs.fwrite</code>	Write binary data to open target computer file
<code>xpctarget.fs.getfilesize</code>	Size of file on target computer
<code>xpctarget.fs.readxpcfile</code>	Interpret raw data from xPC Target file format
<code>xpctarget.fs.removefile</code>	Remove file from target computer
<code>xpctarget.fs.selectdrive</code>	Select target computer drive
<code>xpctarget.fsbased.cd</code>	Change folder on target computer
<code>xpctarget.fsbased.dir</code>	List contents of current folder on target computer
<code>xpctarget.fsbased.mkdir</code>	Make folder on target computer
<code>xpctarget.fsbased.pwd</code>	Current folder path of target computer
<code>xpctarget.fsbased.rmdir</code>	Remove folder from target computer
<code>xpctarget.ftp</code>	Create file transfer protocol (FTP) object
<code>xpctarget.ftp.get (ftp)</code>	Retrieve copy of requested file from target computer
<code>xpctarget.ftp.put</code>	Copy file from host computer to target computer

# Functions

---

# fc422mexcalcbits

---

**Purpose** Calculate parameter values for Fastcom 422/2-PCI board

**Syntax** MATLAB command line

```
[a b ] = fc422mexcalcbits(frequency)
[a b df] = fc422mexcalcbits(frequency)
```

**Arguments** frequency Desired baud rate for the board

[a b] = fc422mexcalcbits(frequency) accepts a baud rate (in units of baud/second) and converts this value into two parameters a b. You must enter these values for the parameter **Clock bits** of the Fastcom 422/2-PCI driver clock. The desired baud rate (frequency) must range between 30e3 and 1.5e6, which is a hardware limitation of the clock circuit.

[a b df] = fc422mexcalcbits(frequency) accepts a baud rate (in units of baud/second) and converts this value into two parameters a b. You must enter these values for the parameter **Clock bits** of the Fastcom 422/2-PCI driver block. The third value, df, indicates the actual baud rate that is created by the generated parameters a b. The clock circuit has limited resolution and is unable to perfectly match an arbitrary frequency. The desired baud rate (frequency) must range between 30e3 and 1.5e6, which is a hardware limitation of the clock circuit.

**Purpose** List environment properties assigned to MATLAB variable

**Syntax** MATLAB command line

```
getxpcenv
```

**Description** Function to list environment properties. This function displays, in the MATLAB Command Window, the property names, the current property values, and the new property values set for the xPC Target environment.

The environment properties define communication between the host computer and target computer, the type of C compiler and its location, and the type of target boot floppy created during the setup process. You can view these properties using the `getxpcenv` function or the xPC Target Explorer. An understanding of the environment properties will help you configure the xPC Target environment.

Environment Property	Description
BootFloppyLocation	Drive name for creation of 3.5-inch target boot disk.
CCompiler	<hr/> <p><b>Note</b></p> <ul style="list-style-type: none"> <li>• CCompiler and CompilerPath will be removed in a future version. Use the <code>xpcgetCC</code> function to configure the compiler instead.</li> <li>• WATCOM compiler support will be removed in a future release. Use a Microsoft compiler instead.</li> </ul> <hr/> <p>Values are 'Watcom' and 'VisualC'. From the xPC Target Explorer window compiler list, select either Watcom or VisualC.</p>
CDBootImageLocation	Location of <code>cdboot.iso</code> file for creation of CD target boot disk.

Environment Property	Description
CompilerPath	<p>Value is a valid compiler root folder. Enter the path where you installed a Watcom C/C++ or Microsoft Visual Studio C/C++ compiler.</p> <p>If the path is invalid or the folder does not contain the compiler, an error message appears when you build a target application.</p>
DOSLoaderLocation	<p>Location of DOSLoader files to boot target computers from devices other than 3.5-inch disk or CD.</p>
EmbeddedOption	<p>Values are 'Disabled' and 'Enabled'. This property is read only.</p> <p>Note that the xPC Target Embedded Option product is enabled only if you purchase an additional license.</p>
EthernetIndex	<p>Value is 'n', where <i>n</i> indicates the index number for the Ethernet card on a target computer. Note that the (n-1)th Ethernet card on the target computer has an index number 'n'. The default index number is 0.</p> <p>If the target computer has multiple Ethernet cards, you must select one of the cards for host-target communication. This option returns the index number of the card selected on the target computer upon booting.</p>



Environment Property	Description
HostTargetComm	<p>Values are 'RS232' and 'TcpIp'.</p> <p>From the xPC Target Explorer window <b>Host target communication</b> list, select either RS232 or TCP/IP.</p> <p>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.</p> <hr/> <p><b>Note</b> RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.</p>
MaxModelSize	<p>BootFloppy and DOSLoader modes ignore this value.</p> <p>Values are '1MB', '4MB', and '16MB'.</p> <p>From the xPC Target Explorer window <b>Maximum model size</b> list, select either 1 MB, 4 MB, or 16 MB. This value is unavailable for BootFloppy or DOSLoader modes.</p> <p>Choosing the maximum model size reserves the specified amount of memory on the target computer for the target application. The remaining memory is used by the kernel and by the heap for data logging.</p> <p>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.</p> <p>Note, you cannot build a 16 MB target application to run in StandAlone mode.</p>

Environment Property	Description
MulticoreSupport	Values are 'off' and 'on'. If your target computer has multicore processors, set this value to 'on' to take advantage of these processors for background tasks. Otherwise, set this value to 'off'.
Name	Target computer name.
NonPentiumSupport	Values are 'off' (default) and 'on'. Set this value to 'off' if your target computer has a 386 or 486 compatible processor. Otherwise, set it to 'on'. If your target computer has a Pentium or higher compatible processor, selecting this check box will slow the performance of your target computer.
RS232Baudrate	<p>Values are '115200', '57600', '38400', '19200', '9600', '4800', '2400', and '1200'.</p> <p>From the <b>Baud rate</b> list, select 115200, 57600, 38400, 19200, 9600, 4800, 2400, or 1200.</p>
RS232HostPort	<p>Values are 'COM1' and 'COM2'.</p> <p>From the xPC Target Explorer window <b>Host port</b> list, select either COM1 or COM2 for the connection on the host computer. The xPC Target software automatically determines the COM port on the target computer.</p> <p>Before you can select an RS-232 port, you need to set the HostTargetComm property to RS232.</p>
SecondaryIDE	Values are 'off' and 'on'. Set this value to 'on' only if you want to use the disks connected to a secondary IDE controller. If you do not have disks connected to the secondary IDE controller, leave this value set to 'off'.

Environment Property	Description
TargetBoot	<p>Values are 'BootFloppy', 'CDBoot', 'DOSLoader', 'NetworkBoot', and 'StandAlone'.</p> <p>From the xPC Target Explorer window target computer configuration pane, select one of the following tabs: <b>Boot Floppy</b>, <b>CD Boot</b>, <b>DOS Loader</b>, <b>Network Boot</b>, or <b>Standalone</b>.</p> <p>If your license file does not include the license for the xPC Target Embedded Option product, your only options are BootFloppy, CDBoot, DOSLoader, and NetworkBoot. With the xPC Target Embedded Option product licensed and installed, you have the additional choice of Standalone.</p>
TargetMACAddress	<p>Physical target computer MAC address from which to accept boot requests when booting within a dedicated network. Enter the address as six pairs of hexadecimal numbers, separated by colons (:).</p>
TargetRAMSizeMB	<p>Values are 'Auto' and 'Manual'.</p> <p>From the xPC Target Explorer window <b>Target RAM size</b> list, select either Auto or Manual. If you select Manual, enter the amount of RAM, in megabytes, installed on the target computer. This property is set by default to Auto.</p> <p><b>Target RAM size</b> defines the total amount of installed RAM in the target computer. This RAM is used for the kernel, target application, data logging, and other functions that use the heap.</p> <p>If <b>Target RAM size</b> is set to Auto, the target application automatically determines the amount of memory up to 64 MB. If the target computer does not contain more than 64 MB of RAM, or you do not want to use more than 64 MB, select Auto. If the target computer has more than 64 MB of RAM, and you want</p>

Environment Property	Description
	<p>to use more than 64 MB, select <b>Manual</b>, and enter the amount of RAM installed in the target computer.</p> <hr/> <p><b>Note</b> The xPC Target kernel can use only 2 GB of memory.</p> <hr/>
TargetScope	<p>Values are 'Disabled' and 'Enabled'.</p> <p>From the xPC Target Explorer window <b>Enable target scope</b> list, select either <b>Enabled</b> or <b>Disabled</b>.</p> <p>The property <b>TargetScope</b> is set by default to <b>Enabled</b>. If you set <b>TargetScope</b> to <b>Disabled</b>, the target computer displays information as text.</p> <p>To use all the features of the target scope, you also need to install a keyboard on the target computer.</p>
TcpIpGateway	<p>Value is 'xxx.xxx.xxx.xxx'.</p> <p>In the xPC Target Explorer window <b>TcpIp gateway address</b> box, enter the IP address for your gateway. This property is set by default to <b>255.255.255.255</b>, which means that a gateway is not used to connect to the target computer.</p> <p>If you communicate with your target computer 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.</p>

Environment Property	Description
TcpIpSubNetMask	<p>Value is 'xxx.xxx.xxx.xxx'.</p> <p>In the xPC Target Explorer window <b>LAN subnet mask address</b> text box, enter the subnet mask of your LAN. Ask your system administrator for this value.</p> <p>For example, your subnet mask could be 255.255.255.0.</p>
TcpIpTargetAddress	<p>Value is 'xxx.xxx.xxx.xxx'.</p> <p>In the xPC Target Explorer window <b>Target PC IP address</b> box, enter a valid IP address for your target computer. Ask your system administrator for this value.</p> <p>For example, 192.168.0.10.</p>
TcpIpTargetBusType	<p>Values are 'PCI' and 'ISA'.</p> <p>From the xPC Target Explorer window <b>TcpIp target bus type</b> list, select either PCI or ISA. This property is set by default to PCI, and determines the bus type of your target computer. You do not need to define a bus type for your host computer, which can be the same or different from the bus type in your target computer.</p> <p>If TcpIpTargetBusType is set to PCI, then the properties TcpIpISAMemPort and TcpIpISAIRQ have no effect on TCP/IP communication.</p> <p>If you are using an ISA bus card, set TcpIpTargetBusType to ISA and enter values for TcpIpISAMemPort and TcpIpISAIRQ.</p>

Environment Property	Description
TcpIpTargetDriver	<p>Values are 'NE2000', 'SMC91C9X', 'I82559', 'RTLANCE', 'R8139', '3C90x', and 'NS83815'.</p> <p>From the xPC Target Explorer window <b>TcpIp target driver</b> list, select NE2000, SMC91C9X, I82559, RTLANCE, R8139, 3C90x, or NS83815.</p>
TcpIpTargetPort	<p>Value is 'xxxxx'.</p> <p>In the xPC Target Explorer window <b>TcpIp target port</b> box, enter a value greater than 20000.</p> <p>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 computer.</p>
TcpIpTargetISAIRQ	<p>Value is 'n', where <i>n</i> is between 4 and 15.</p> <p>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.</p> <p>On your ISA bus card, assign an IRQ and I/O-port base address by moving the jumpers on the card.</p> <p>Set the IRQ to 5, 10, or 11. If one of these hardware settings leads to a conflict in your target computer, choose another IRQ and make the corresponding changes to your jumper settings.</p>

Environment Property	Description
TcpIpTargetISAMemPort	<p>Value is '0xnxxxx'.</p> <p>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.</p> <p>On your ISA bus card, assign an IRQ and I/O port base address by moving the jumpers on the card.</p> <p>Set the I/O port base address to around 0x300. If one of these hardware settings leads to a conflict in your target computer, choose another I/O port base address and make the corresponding changes to your jumper settings.</p>
Version	xPC Target version number. Read only.

## 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

setxpcenv | xpcbootdisk

# getxpcinfo

---

**Purpose** Retrieve diagnostic information to help troubleshoot configuration issues

**Syntax** MATLAB command line

```
getxpcinfo  
getxpcinfo('-a')
```

**Arguments** '-a' Appends diagnostic information to an existing `xpcinfo.txt` file. If one does not exist, this function creates the file in the current folder.

**Description** `getxpcinfo` returns diagnostic information for troubleshooting xPC Target configuration issues. This function generates and saves the information in the `xpcinfo.txt` file, in the current folder. If the file `xpcinfo.txt` already exists, this function overwrites it with the new information.

`getxpcinfo('-a')` appends the diagnostic information to the `xpcinfo.txt` file, in the current folder. If the file `xpcinfo.txt` does not exist, this function creates it.

You can send the file `xpcinfo.txt` to MathWorks Technical Support for evaluation and guidance. To create this file, you must have write permission for the current folder.

## Warning

**The file `xpcinfo.txt` might contain information sensitive to your organization. Review the contents of this file before sending to MathWorks.**



---

<b>Purpose</b>	Convert string-based MAC address to vector-based one
<b>Syntax</b>	<b>MATLAB command line</b>  <code>macaddr('MAC address')</code>
<b>Argument</b>	'MAC address'      String-based MAC address to be converted.
<b>Description</b>	The <code>macaddr</code> function converts a string-based MAC address to a vector-based MAC address. The string-based MAC address should be a string comprised of six colon-delimited fields of two-digit hexadecimal numbers.
<b>Examples</b>	<code>macaddr('01:23:45:67:89:ab')</code>  <code>ans =</code>  1    35    69   103   137   171
<b>How To</b>	<ul style="list-style-type: none"><li>• “Model-Based Ethernet Communications Support”</li><li>• <i>xPC Target I/O Reference</i></li></ul>

# setxpcenv

---

**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 to enter new values for environment properties. If the new value is different from the current value, the property is marked as having a new value.

The environment properties define communication between the host computer and target computer, the type of C compiler and its location, and the type of target boot floppy created during the setup process. With the exception of the `Version` property, you can set environment properties using the `setxpcenv` function or the xPC Target Explorer window, accessed via the `xpcexplr` function. An understanding of the environment properties will help you configure the xPC Target environment.

Environment Property	Description
BootFloppyLocation	Drive name for creation of 3.5-inch target boot disk.
CCompiler	<hr/> <p><b>Note</b></p> <ul style="list-style-type: none"> <li>• CCompiler and CompilerPath will be removed in a future version. Use the xpcsetCC function to configure the compiler instead.</li> <li>• WATCOM compiler support will be removed in a future release. Use a Microsoft compiler instead.</li> </ul> <hr/> <p>Values are 'Watcom' and 'VisualC'. From the xPC Target Explorer window compiler list, select either Watcom or VisualC.</p>
CDBootImageLocation	Location of cdboot.iso file for creation of CD target boot disk.
CompilerPath	<p>Value is a valid compiler root folder. Enter the path where you installed a Watcom C/C++ or Microsoft Visual Studio C/C++ compiler.</p> <p>If the path is invalid or the folder does not contain the compiler, an error message appears when you build a target application.</p>
DOSLoaderLocation	Location of DOSLoader files to boot target computers from devices other than 3.5-inch disk or CD.
EmbeddedOption	<p>Values are 'Disabled' and 'Enabled'. This property is read only.</p> <p>Note that the xPC Target Embedded Option product is enabled only if you purchase an additional license.</p>

Environment Property	Description
EthernetIndex	<p>Value is 'n', where <i>n</i> indicates the index number for the Ethernet card on a target computer. Note that the (n-1)th Ethernet card on the target computer has an index number 'n'. The default index number is 0.</p> <p>If the target computer has multiple Ethernet cards, you must select one card for host-target communication. This option returns the index number of the card selected on the target computer upon booting.</p>
HostTargetComm	<p>Values are 'RS232' and 'TcpIp'.</p> <p>From the xPC Target Explorer window <b>Host target communication</b> list, select either RS232 or TCP/IP.</p> <p>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.</p> <hr/> <p><b>Note</b> RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.</p> <hr/>
MaxModelSize	<p>BootFloppy and DOSLoader modes ignore this value.</p> <p>Values are '1MB', '4MB', and '16MB'.</p> <p>From the xPC Target Explorer window <b>Maximum model size</b> list, select either 1 MB, 4 MB, or 16 MB. This value is unavailable for BootFloppy or DOSLoader modes.</p> <p>Choosing the maximum model size reserves the specified amount of memory on the target computer for the target application. The remaining memory is used by the kernel and by the heap for data logging.</p>

Environment Property	Description
	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.
MulticoreSupport	Values are 'off' and 'on'. If your target computer has multicore processors, set this value to 'on' to take advantage of these processors for background tasks. Otherwise, set this value to 'off'.
NonPentiumSupport	Values are 'off' (default) and 'on'. Set this value to 'off' if your target computer has a 386 or 486 compatible processor. Otherwise, set it to 'on'. If your target computer has a Pentium or higher compatible processor, selecting this check box will slow the performance of your target computer.
RS232Baudrate	Values are '115200', '57600', '38400', '19200', '9600', '4800', '2400', and '1200'.  From the <b>Baud rate</b> list, select 115200, 57600, 38400, 19200, 9600, 4800, 2400, or 1200.
RS232HostPort	Values are 'COM1' and 'COM2'.  From the xPC Target Explorer window <b>Host port</b> list, select either COM1 or COM2 for the connection on the host computer. The xPC Target software automatically determines the COM port on the target computer.  Before you can select an RS-232 port, you need to set the HostTargetComm property to RS232.
SecondaryIDE	Values are 'off' and 'on'. Set this value to 'on' only if you want to use the disks connected to a secondary IDE controller. If you do not have disks connected to the secondary IDE controller, leave this value set to 'off'.

Environment Property	Description
TargetBoot	<p>Values are 'BootFloppy', 'CDBoot', 'DOSLoader', 'NetworkBoot', and 'StandAlone'.</p> <p>From the xPC Target Explorer window target computer configuration pane, select one of the following tabs: <b>Boot Floppy</b>, <b>CD Boot</b>, <b>DOS Loader</b>, <b>Network Boot</b>, or <b>Standalone</b>.</p> <p>If your license file does not include the license for the xPC Target Embedded Option product, your only options are <b>BootFloppy</b>, <b>CDBoot</b>, <b>DOSLoader</b>, and <b>NetworkBoot</b>. With the xPC Target Embedded Option product licensed and installed, you have the additional choice of <b>Standalone</b>.</p>
TargetMACAddress	<p>Physical target computer MAC address from which to accept boot requests when booting within a dedicated network. Enter the address as six pairs of hexadecimal numbers, separated by colons (:).</p>
TargetRAMSizeMB	<p>Values are 'Auto' and 'Manual'.</p> <p>From the xPC Target Explorer window <b>Target RAM size</b> list, select either <b>Auto</b> or <b>Manual</b>. If you select <b>Manual</b>, enter the amount of RAM, in megabytes, installed on the target computer. This property is set by default to <b>Auto</b>.</p> <p><b>Target RAM size</b> defines the total amount of installed RAM in the target computer. This RAM is used for the kernel, target application, data logging, and other functions that use the heap.</p> <p>If <b>Target RAM size</b> is set to <b>Auto</b>, the target application automatically determines the amount of memory up to 64 MB. If the target computer does not contain more than 64 MB of RAM, or you do not want to use more than 64 MB, select <b>Auto</b>. If the target computer has more than 64 MB of RAM, and you want to use more than 64 MB, select <b>Manual</b>, and enter the amount of RAM installed in the target computer.</p>

Environment Property	Description
	<p><b>Note</b> The xPC Target kernel can use only 2 GB of memory.</p>
TargetScope	<p>Values are 'Disabled' and 'Enabled'.</p> <p>From the xPC Target Explorer window <b>Enable target scope</b> list, select either Enabled or Disabled.</p> <p>The property TargetScope is set by default to Enabled. If you set TargetScope to Disabled, the target computer displays information as text.</p> <p>To use all the features of the target scope, you also need to install a keyboard on the target computer.</p>
TcpIpGateway	<p>Value is 'xxx.xxx.xxx.xxx'.</p> <p>In the xPC Target Explorer window <b>TcpIp gateway address</b> 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 computer.</p> <p>If you communicate with your target computer 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.</p>
TcpIpSubNetMask	<p>Value is 'xxx.xxx.xxx.xxx'.</p> <p>In the xPC Target Explorer window <b>LAN subnet mask address</b> text box, enter the subnet mask of your LAN. Ask your system administrator for this value.</p> <p>For example, your subnet mask could be 255.255.255.0.</p>

Environment Property	Description
TcpIpTargetAddress	<p>Value is 'xxx.xxx.xxx.xxx'.</p> <p>In the xPC Target Explorer window <b>Target PC IP address</b> box, enter a valid IP address for your target computer. Ask your system administrator for this value.</p> <p>For example, 192.168.0.10.</p>
TcpIpTargetBusType	<p>Values are 'PCI' and 'ISA'.</p> <p>From the xPC Target Explorer window <b>TcpIp target bus type</b> list, select either PCI or ISA. This property is set by default to PCI, and determines the bus type of your target computer. You do not need to define a bus type for your host computer, which can be the same or different from the bus type in your target computer.</p> <p>If TcpIpTargetBusType is set to PCI, then the properties TcpIpISAMemPort and TcpIpISAIRQ have no effect on TCP/IP communication.</p> <p>If you are using an ISA bus card, set TcpIpTargetBusType to ISA and enter values for TcpIpISAMemPort and TcpIpISAIRQ.</p>
TcpIpTargetDriver	<p>Values are 'NE2000', 'SMC91C9X', 'I82559', 'RTLANCE', 'R8139', '3C90x', and 'NS83815'.</p> <p>From the xPC Target Explorer window <b>TcpIp target driver</b> list, select NE2000, SMC91C9X, I82559, RTLANCE, R8139, 3C90x, or NS83815.</p>



Environment Property	Description
TcpIpTargetPort	<p>Value is 'xxxxx'.</p> <p>In the xPC Target Explorer window <b>TcpIp target port</b> box, enter a value greater than 20000.</p> <p>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 computer.</p>
TcpIpTargetISAIRQ	<p>Value is 'n', where <i>n</i> is between 4 and 15.</p> <p>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.</p> <p>On your ISA bus card, assign an IRQ and I/O-port base address by moving the jumpers on the card.</p> <p>Set the IRQ to 5, 10, or 11. If one of these hardware settings leads to a conflict in your target computer, choose another IRQ and make the corresponding changes to your jumper settings.</p>
TcpIpTargetISAMemPort	<p>Value is '0xnxxx'.</p> <p>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.</p> <p>On your ISA bus card, assign an IRQ and I/O port base address by moving the jumpers on the card.</p> <p>Set the I/O port base address to around 0x300. If one of these hardware settings leads to a conflict in your target</p>

# setxpcenv

---

Environment Property	Description
	computer, choose another I/O port base address and make the corresponding changes to your jumper settings.
Version	xPC Target version number. Read only.

The function `setxpcenv` works similarly to the `set` function of the MATLAB Handle Graphics® system. Call the function `setxpcenv` 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.

```
setxpcenv
```

Change the serial communication port of the host computer to COM2.

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

## See Also

`getxpcenv` | `xpcbootdisk`

## How To

- “Changing Environment Properties with xPC Target Explorer” on page 4-16
- “Changing Environment Properties with a Command-Line Interface for Single Target Computer Systems” on page 4-20

**Purpose** xPC Target benchmark

**Syntax** MATLAB command line

```
xpcbench(model)
xpcbench(model,P1)
xpcbench(model,P1,P2)
xpcbench(model,P1,P2,P3)
```

**Arguments**

Argument	Value	Description
model	'minimal'	Benchmark the default minimal model.
	'f14'	Benchmark the default f14 model (one f14 system).
	'f14*5'	Benchmark the default f14*5 model (five copies of the f14 system).
	'f14*10'	Benchmark the default f14*10 model (ten copies of the f14 system).
	'f14*25'	Benchmark the default f14*25 model (25 copies of the f14 system).
	'this'	Benchmark all five default models (the default minimal model plus all four default f14 models).

Argument	Value	Description
	' <i>usermdl</i> '	Benchmark your model, <i>usermdl.mdl</i> .
Up to three optional arguments: P1, P2, and P3	'-reboot'	Reboot the target computer after testing each model.
	'-cleanup'	Delete build files after running benchmarks.
	'-verbose'	Plot and display results in the MATLAB Command Window.

## Description

xpcbench benchmarks the real-time execution performance of xPC Target applications on your target computer and compares the result with prestored benchmark results from other computers.

The prestored target computer benchmark results of five models (applications) are provided, each model compiled using a sampling of the currently supported compilers (see [http://www.mathworks.com/support/compilers/current\\_release/](http://www.mathworks.com/support/compilers/current_release/)). Compare these results with those for your target computer. Results are labeled by CPU type, CPU clock rate, and the compiler used to compile the application.

The five benchmark models are:

**Benchmark**

**Description**

minimal

Based on a minimal model consisting of just three blocks (Constant, Gain, Termination). This model has neither continuous nor discrete states. The result of this benchmark gives an impression about the target computer interrupt latencies.

f14

Based on the standard Simulink example model f14. Type `f14` in the MATLAB Command Window to open the model and view the model (62 blocks, 10 continuous states).

f14\*5

Five f14 systems modeled in subsystems (310 blocks, 50 continuous states). This benchmark is five times more demanding than benchmark F14.

f14\*10

Ten f14 systems (620 blocks, 100 continuous states).

f14\*25

25 f14 systems (1550 blocks, 250 continuous states).

xpcbench without an argument displays two plot figures, each containing different representations of the prestored target computer benchmark results. The first plot lists, for each target computer tested, the smallest achievable sample time for the five benchmarks, in microseconds. The second plot contains a bar graph of all computers, ranked by relative performance.

---

## Note

- The prestored benchmark results were collected with **Multicore CPU support** disabled. This allows for direct comparison with previous release results.
  - A sample time is achievable if any smaller sample time causes CPU overload.
- 

`xpcbench(model)` benchmarks your target computer using argument `model`. You can specify:

- All five default benchmarks ('this')
- One of the five default benchmarks ('minimal', 'f14', 'f14\*5', 'f14\*10', 'f14\*25')
- Your model ('*usermdl*')

Before you run `xpcbench`, you must connect the host machine to the target computer and run the xPC Target test, `xpctest`, with no failures.

Benchmark execution can take several minutes to complete, including:

- 1 Generating the benchmark models
- 2 Building and downloading the xPC Target applications
- 3 Searching for the smallest achievable sample time
- 4 Displaying the performance results in the MATLAB Command Window, along with the prestored results for the other target computers

`res = xpcbench` returns the prestored benchmark results in a structure array with fields:

Field Name	Contents
<i>Machine</i>	Target computer information string containing CPU type, CPU speed, compiler
<i>BenchResults</i>	Target computer benchmark performance for the five default models 'minimal', 'f14', 'f14*5', 'f14*10', 'f14*25'
<i>Desc</i>	Target computer descriptor string containing machine type, RAM size, cache size

`res = xpcbench(model)` returns the benchmark results for `model` in a structure with fields:

Field Name	Contents
<i>Name</i>	Name of <code>model</code>
<i>nBlocks</i>	Number of blocks in <code>model</code>
<i>BuildTime</i>	Elapsed time in seconds to build <code>model</code>
<i>BenchTime</i>	Elapsed time in seconds to run benchmark for <code>model</code>
<i>Tsmin</i>	Minimal achievable sample time in seconds for <code>model</code>

## Examples

### xpcbench

Prestored benchmark results showing what to expect of representative processors.

Boot the target computer using your chosen method.

Connect it to the host computer.

# xpcbench

---

```
xptest
```

```
res = xpcbench;  
res(1)
```

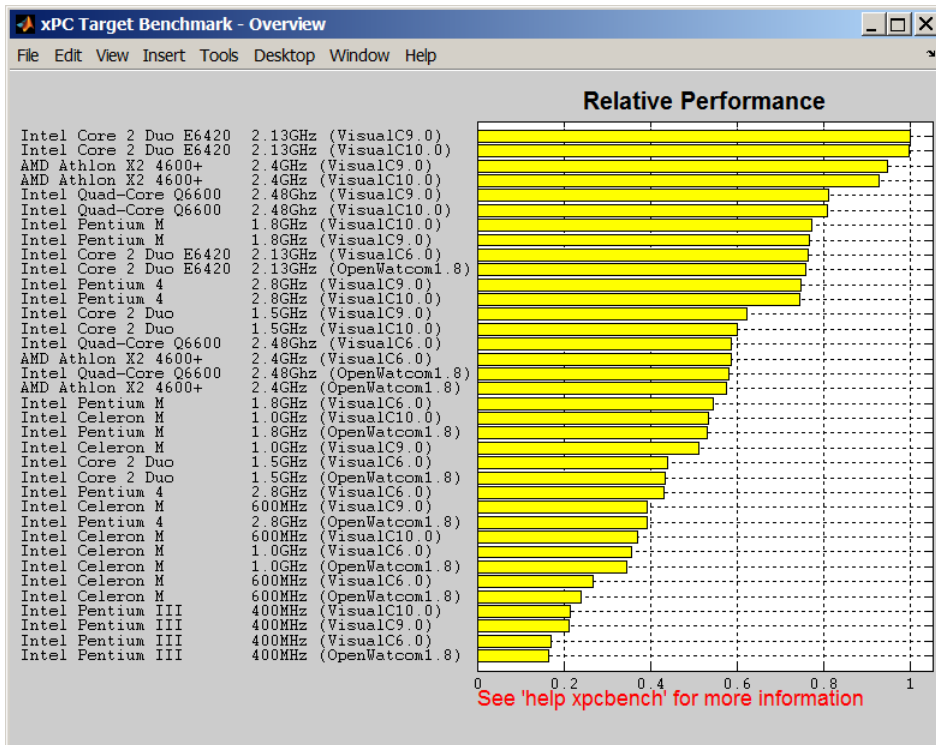
```
ans =
```

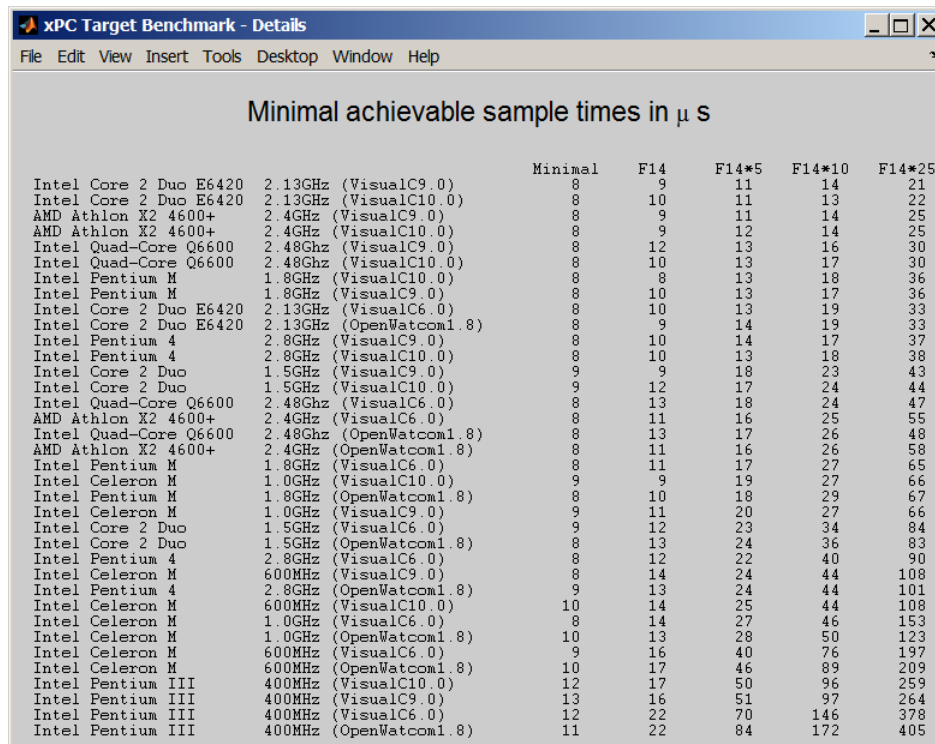
```
Machine: 'Intel Celeron M 600MHz (VisualC10.0)'  
BenchResults: [1.0000e-05 1.4000e-05 2.5000e-05  
4.4000e-05 1.0800e-04]  
Desc: [1x70 char]
```

The array contains benchmark results for each processor type, in arbitrary order.

```
xpcbench
```







Minimal achievable sample times in  $\mu$  s

		Minimal	F14	F14*5	F14*10	F14*25
Intel Core 2 Duo E6420	2.13GHz (VisualC9.0)	8	9	11	14	21
Intel Core 2 Duo E6420	2.13GHz (VisualC10.0)	8	10	11	13	22
AMD Athlon X2 4600+	2.4GHz (VisualC9.0)	8	9	11	14	25
AMD Athlon X2 4600+	2.4GHz (VisualC10.0)	8	9	12	14	25
Intel Quad-Core Q6600	2.48Ghz (VisualC9.0)	8	12	13	16	30
Intel Quad-Core Q6600	2.48Ghz (VisualC10.0)	8	10	13	17	30
Intel Pentium M	1.8GHz (VisualC10.0)	8	8	13	18	36
Intel Pentium M	1.8GHz (VisualC9.0)	8	10	13	17	36
Intel Core 2 Duo E6420	2.13GHz (VisualC6.0)	8	10	13	19	33
Intel Core 2 Duo E6420	2.13GHz (OpenWatcom1.8)	8	9	14	19	33
Intel Pentium 4	2.8GHz (VisualC9.0)	8	10	14	17	37
Intel Pentium 4	2.8GHz (VisualC10.0)	8	10	13	18	38
Intel Core 2 Duo	1.5GHz (VisualC9.0)	9	9	18	23	43
Intel Core 2 Duo	1.5GHz (VisualC10.0)	9	12	17	24	44
Intel Quad-Core Q6600	2.48GHz (VisualC6.0)	8	13	18	24	47
AMD Athlon X2 4600+	2.4GHz (VisualC6.0)	8	11	16	25	55
Intel Quad-Core Q6600	2.48GHz (OpenWatcom1.8)	8	13	17	26	48
AMD Athlon X2 4600+	2.4GHz (OpenWatcom1.8)	8	11	16	26	58
Intel Pentium M	1.8GHz (VisualC6.0)	8	11	17	27	65
Intel Celeron M	1.0GHz (VisualC10.0)	9	9	19	27	66
Intel Pentium M	1.8GHz (OpenWatcom1.8)	8	10	18	29	67
Intel Celeron M	1.0GHz (VisualC9.0)	9	11	20	27	66
Intel Core 2 Duo	1.5GHz (VisualC6.0)	9	12	23	34	84
Intel Core 2 Duo	1.5GHz (OpenWatcom1.8)	8	13	24	36	83
Intel Pentium 4	2.8GHz (VisualC6.0)	8	12	22	40	90
Intel Celeron M	600MHz (VisualC9.0)	8	14	24	44	108
Intel Pentium 4	2.8GHz (OpenWatcom1.8)	9	13	24	44	101
Intel Celeron M	600MHz (VisualC10.0)	10	14	25	44	108
Intel Celeron M	1.0GHz (VisualC6.0)	8	14	27	46	153
Intel Celeron M	1.0GHz (OpenWatcom1.8)	10	13	28	50	123
Intel Celeron M	600MHz (VisualC6.0)	9	16	40	76	197
Intel Celeron M	600MHz (OpenWatcom1.8)	10	17	46	89	209
Intel Pentium III	400MHz (VisualC10.0)	12	17	50	96	259
Intel Pentium III	400MHz (VisualC9.0)	13	16	51	97	264
Intel Pentium III	400MHz (VisualC6.0)	12	22	70	146	378
Intel Pentium III	400MHz (OpenWatcom1.8)	11	22	84	172	405

## xpcbench('this')

Benchmark using the five default models.

Boot the target computer using your chosen method.

Connect it to the host computer.

```
xptest
```

```
res = xpcbench('this');
```

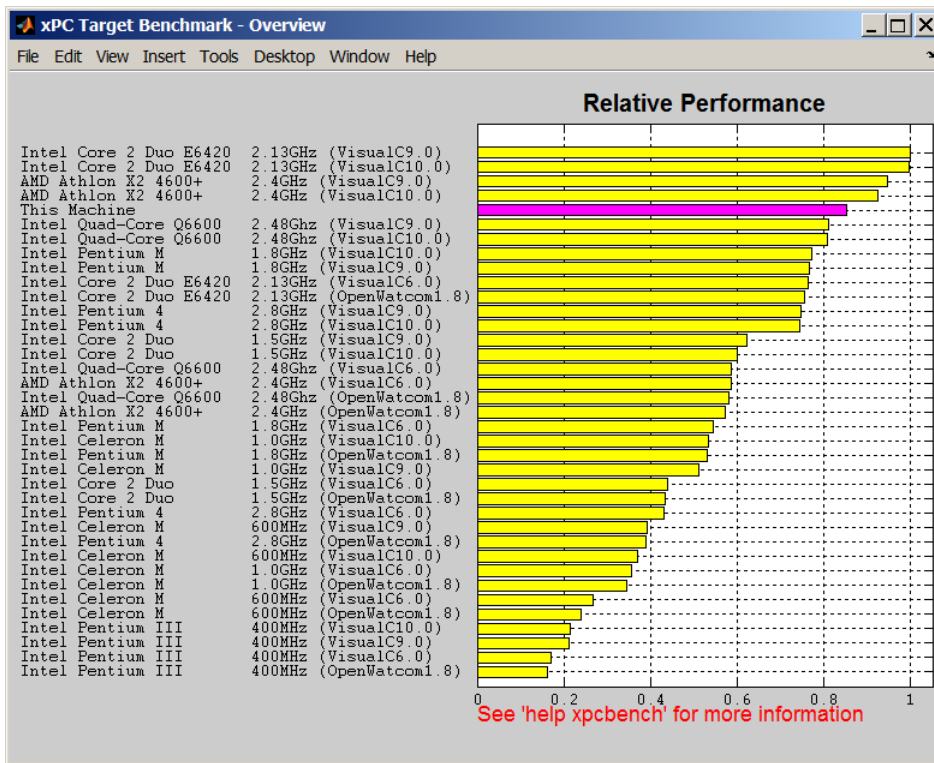
```
res(1)
```

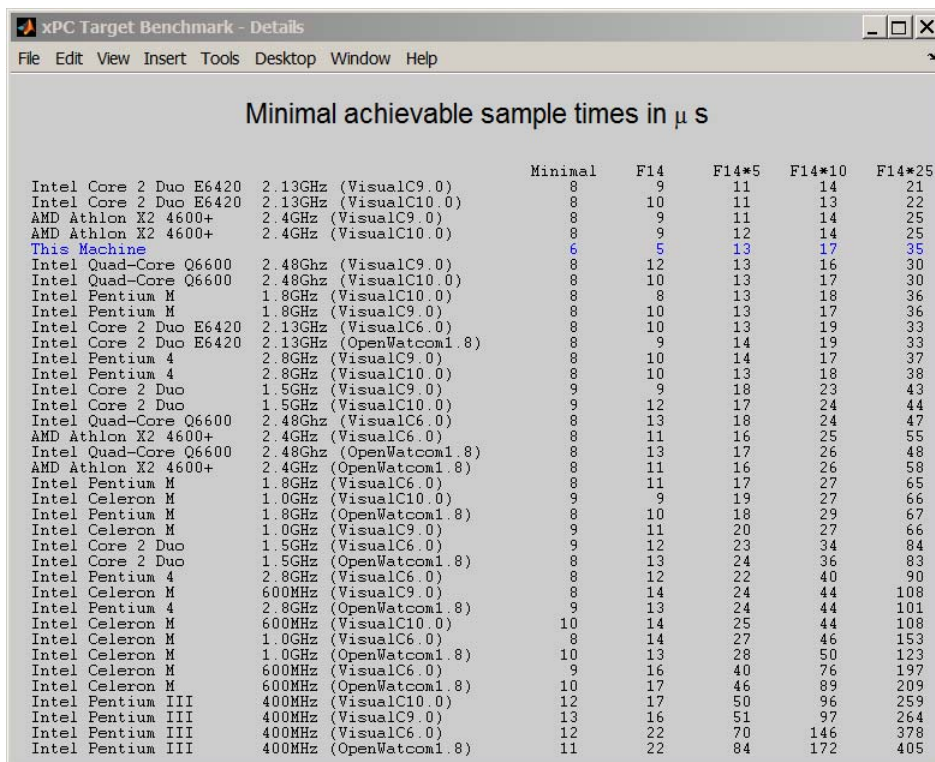
```
ans =
```

```

Name: 'Minimal'
nBlocks: 3
BuildTime: 28.2558
BenchTime: 23.3845
TsmIn: 8.3770e-06
    
```

xpcbench('this')





Minimal achievable sample times in  $\mu$  s

		Minimal	F14	F14*5	F14*10	F14*25
Intel Core 2 Duo E6420	2.13GHz (VisualC9.0)	8	9	11	14	21
Intel Core 2 Duo E6420	2.13GHz (VisualC10.0)	8	10	11	13	22
AMD Athlon X2 4600+	2.4GHz (VisualC9.0)	8	9	11	14	25
AMD Athlon X2 4600+	2.4GHz (VisualC10.0)	8	9	12	14	25
<b>This Machine</b>		<b>6</b>	<b>5</b>	<b>13</b>	<b>17</b>	<b>35</b>
Intel Quad-Core Q6600	2.48Ghz (VisualC9.0)	8	12	13	16	30
Intel Quad-Core Q6600	2.48Ghz (VisualC10.0)	8	10	13	17	30
Intel Pentium M	1.8GHz (VisualC10.0)	8	8	13	18	36
Intel Pentium M	1.8GHz (VisualC9.0)	8	10	13	17	36
Intel Core 2 Duo E6420	2.13GHz (VisualC6.0)	8	10	13	19	33
Intel Core 2 Duo E6420	2.13GHz (OpenWatcom1.8)	8	9	14	19	33
Intel Pentium 4	2.8GHz (VisualC9.0)	8	10	14	17	37
Intel Pentium 4	2.8GHz (VisualC10.0)	8	10	13	18	38
Intel Core 2 Duo	1.5GHz (VisualC9.0)	9	9	18	23	43
Intel Core 2 Duo	1.5GHz (VisualC10.0)	9	12	17	24	44
Intel Quad-Core Q6600	2.48Ghz (VisualC6.0)	8	13	18	24	47
AMD Athlon X2 4600+	2.4GHz (VisualC6.0)	8	11	16	25	55
Intel Quad-Core Q6600	2.48Ghz (OpenWatcom1.8)	8	13	17	26	48
AMD Athlon X2 4600+	2.4GHz (OpenWatcom1.8)	8	11	16	26	58
Intel Pentium M	1.8GHz (VisualC6.0)	8	11	17	27	65
Intel Celeron M	1.0GHz (VisualC10.0)	9	9	19	27	66
Intel Pentium M	1.8GHz (OpenWatcom1.8)	8	10	18	29	67
Intel Celeron M	1.0GHz (VisualC9.0)	9	11	20	27	66
Intel Core 2 Duo	1.5GHz (VisualC6.0)	9	12	23	34	84
Intel Core 2 Duo	1.5GHz (OpenWatcom1.8)	8	13	24	36	83
Intel Pentium 4	2.8GHz (VisualC6.0)	8	12	22	40	90
Intel Celeron M	600MHz (VisualC9.0)	8	14	24	44	108
Intel Pentium 4	2.8GHz (OpenWatcom1.8)	9	13	24	44	101
Intel Celeron M	600MHz (VisualC10.0)	10	14	25	44	108
Intel Celeron M	1.0GHz (VisualC6.0)	8	14	27	46	153
Intel Celeron M	1.0GHz (OpenWatcom1.8)	10	13	28	50	123
Intel Celeron M	600MHz (VisualC6.0)	9	16	40	76	197
Intel Celeron M	600MHz (OpenWatcom1.8)	10	17	46	89	209
Intel Pentium III	400MHz (VisualC10.0)	12	17	50	96	259
Intel Pentium III	400MHz (VisualC9.0)	13	16	51	97	264
Intel Pentium III	400MHz (VisualC6.0)	12	22	70	146	378
Intel Pentium III	400MHz (OpenWatcom1.8)	11	22	84	172	405

## xpcbench('f14\*5')

Benchmark using model f14\*5 only.

Boot the target computer using your chosen method.

Connect it to the host computer.

```
xpctest
```

```
res = xpcbench('f14*5')
```

```
res
```

```
res =
```

```
Name: 'F14*5'  
nBlocks: 310  
BuildTime: 32.8377  
BenchTime: 29.7613  
TsmIn: 1.3148e-05
```

```
xpcbench('f14*5')
```

```
Benchmark results for model:           F14*5  
Number of blocks in model:           310  
Elapsed time for model build (sec):    38.3  
Elapsed time for model benchmark (sec): 29.7  
Minimal achievable sample time (microsec): 14.1
```

### **xpcbench('this', '-reboot')**

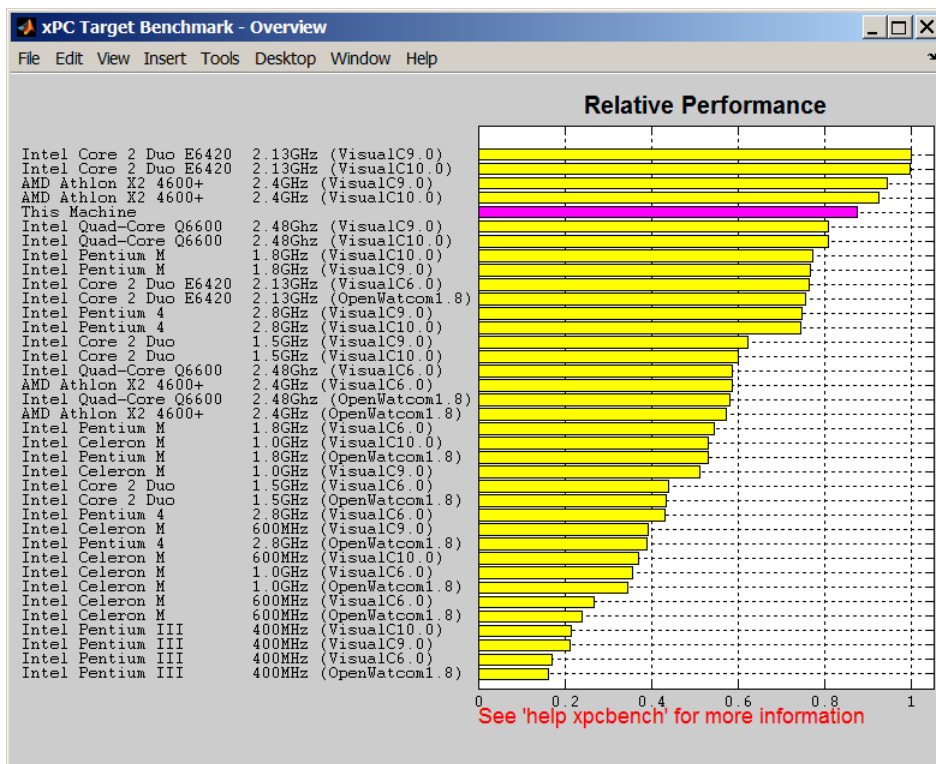
Benchmark the target computer using the five default models, rebooting after each test.

Boot the target computer using your chosen method.

Connect it to the host computer.

```
xptest
```

```
xpcbench('this', '-reboot')
```



The screenshot shows a window titled "xPC Target Benchmark - Details" with a menu bar (File, Edit, View, Insert, Tools, Desktop, Window, Help). The main content area displays a table of processor configurations and their corresponding minimal achievable sample times in microseconds (μs) for five different test cases: Minimal, F14, F14\*5, F14\*10, and F14\*25. The table lists various Intel and AMD processors, including Core 2 Duo, Athlon X2, Quad-Core, Pentium M, and Celeron M, with their respective clock speeds and operating systems. The "This Machine" row is highlighted in blue, showing a minimal time of 5 μs and a 14x multiplier time of 35 μs.

		Minimal	F14	F14*5	F14*10	F14*25
Intel Core 2 Duo E6420	2.13GHz (VisualC9.0)	8	9	11	14	21
Intel Core 2 Duo E6420	2.13GHz (VisualC10.0)	8	10	11	13	22
AMD Athlon X2 4600+	2.4GHz (VisualC9.0)	8	9	11	14	25
AMD Athlon X2 4600+	2.4GHz (VisualC10.0)	8	9	12	14	25
<b>This Machine</b>		<b>5</b>	<b>6</b>	<b>13</b>	<b>17</b>	<b>35</b>
Intel Quad-Core Q6600	2.48GHz (VisualC9.0)	8	12	13	16	30
Intel Quad-Core Q6600	2.48GHz (VisualC10.0)	8	10	13	17	30
Intel Pentium M	1.8GHz (VisualC10.0)	8	8	13	18	36
Intel Pentium M	1.8GHz (VisualC9.0)	8	10	13	17	36
Intel Core 2 Duo E6420	2.13GHz (VisualC6.0)	8	10	13	19	33
Intel Core 2 Duo E6420	2.13GHz (OpenWatcom1.8)	8	9	14	19	33
Intel Pentium 4	2.8GHz (VisualC9.0)	8	10	14	17	37
Intel Pentium 4	2.8GHz (VisualC10.0)	8	10	13	18	38
Intel Core 2 Duo	1.5GHz (VisualC9.0)	9	9	18	23	43
Intel Core 2 Duo	1.5GHz (VisualC10.0)	9	12	17	24	44
Intel Quad-Core Q6600	2.48GHz (VisualC6.0)	8	13	18	24	47
AMD Athlon X2 4600+	2.4GHz (VisualC6.0)	8	11	16	25	55
Intel Quad-Core Q6600	2.48GHz (OpenWatcom1.8)	8	13	17	26	48
AMD Athlon X2 4600+	2.4GHz (OpenWatcom1.8)	8	11	16	26	58
Intel Pentium M	1.8GHz (VisualC6.0)	8	11	17	27	65
Intel Celeron M	1.0GHz (VisualC10.0)	9	9	19	27	66
Intel Pentium M	1.8GHz (OpenWatcom1.8)	8	10	18	29	67
Intel Celeron M	1.0GHz (VisualC9.0)	9	11	20	27	66
Intel Core 2 Duo	1.5GHz (VisualC6.0)	9	12	23	34	84
Intel Core 2 Duo	1.5GHz (OpenWatcom1.8)	8	13	24	36	83
Intel Pentium 4	2.8GHz (VisualC6.0)	8	12	22	40	90
Intel Celeron M	600MHz (VisualC9.0)	8	14	24	44	108
Intel Pentium 4	2.8GHz (OpenWatcom1.8)	9	13	24	44	101
Intel Celeron M	600MHz (VisualC10.0)	10	14	25	44	108
Intel Celeron M	1.0GHz (VisualC6.0)	8	14	27	46	153
Intel Celeron M	1.0GHz (OpenWatcom1.8)	10	13	28	50	123
Intel Celeron M	600MHz (VisualC6.0)	9	16	40	76	197
Intel Celeron M	600MHz (OpenWatcom1.8)	10	17	46	89	209
Intel Pentium III	400MHz (VisualC10.0)	12	17	50	96	259
Intel Pentium III	400MHz (VisualC9.0)	13	16	51	97	264
Intel Pentium III	400MHz (VisualC6.0)	12	22	70	146	378
Intel Pentium III	400MHz (OpenWatcom1.8)	11	22	84	172	405

**xpcbench('xpcosc','-cleanup','-verbose');**

Benchmark the target computer using model xpcosc.mdl, delete the build files when done, and display full results in the MATLAB Command Window.

Boot the target computer using your chosen method.

Connect it to the host computer.

xpctest

xpcbench('xpcosc','-cleanup','-verbose');

# xpcbench

---

```
Benchmark results for model:           xpcosc
Number of blocks in model:             287
Elapsed time for model build (sec):    28.7
Elapsed time for model benchmark (sec): 26.1
Minimal achievable sample time (microsec): 5.5
```

**res = xpcbench('this','-reboot','-cleanup','-verbose');**

Benchmark the target computer using the five default models, reboot after each test, store the results in *res*, delete the build files when done, and display full results in the MATLAB Command Window.

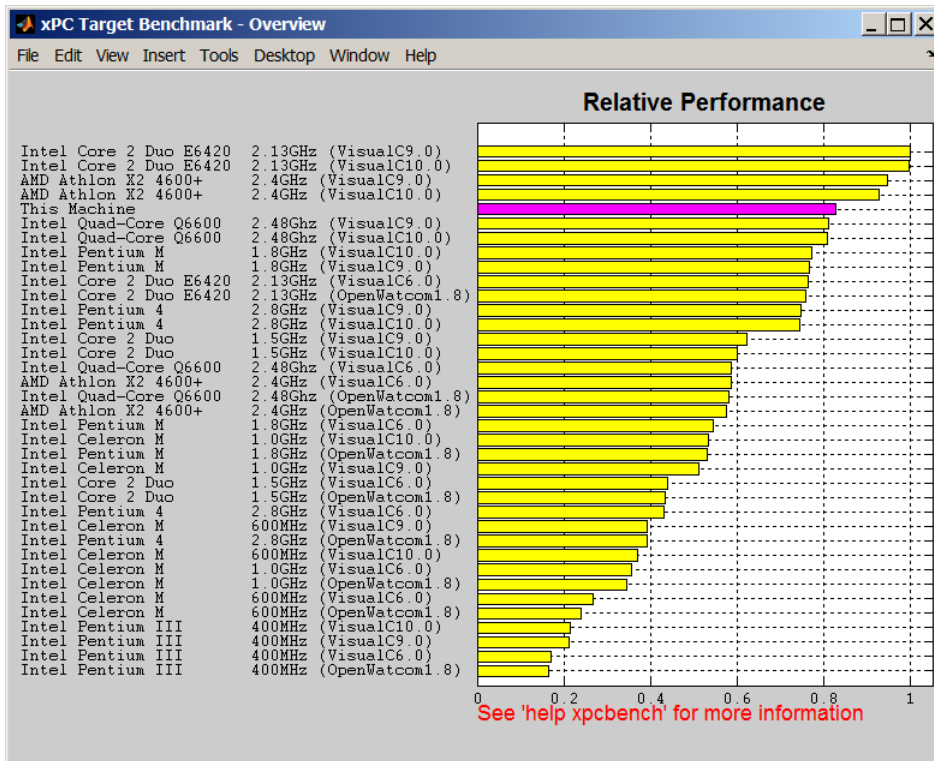
Boot the target computer using your chosen method.

Connect it to the host computer.

```
xptest
```

```
res = xpcbench('this','-reboot','-cleanup','-verbose');
```





xPC Target Benchmark - Details

File Edit View Insert Tools Desktop Window Help

Minimal achievable sample times in  $\mu$  s

		Minimal	F14	F14*5	F14*10	F14*25
Intel Core 2 Duo E6420	2.13GHz (VisualC9.0)	8	9	11	14	21
Intel Core 2 Duo E6420	2.13GHz (VisualC10.0)	8	10	11	13	22
AMD Athlon X2 4600+	2.4GHz (VisualC9.0)	8	9	11	14	25
AMD Athlon X2 4600+	2.4GHz (VisualC10.0)	8	9	12	14	25
<u>This Machine</u>		<b>5</b>	<b>11</b>	<b>13</b>	<b>16</b>	<b>35</b>
Intel Quad-Core Q6600	2.48Ghz (VisualC9.0)	8	12	13	16	30
Intel Quad-Core Q6600	2.48Ghz (VisualC10.0)	8	10	13	17	30
Intel Pentium M	1.8GHz (VisualC10.0)	8	8	13	18	36
Intel Pentium M	1.8GHz (VisualC9.0)	8	10	13	17	36
Intel Core 2 Duo E6420	2.13GHz (VisualC6.0)	8	10	13	19	33
Intel Core 2 Duo E6420	2.13GHz (OpenWatcom1.8)	8	9	14	19	33
Intel Pentium 4	2.8GHz (VisualC9.0)	8	10	14	17	37
Intel Pentium 4	2.8GHz (VisualC10.0)	8	10	13	18	38
Intel Core 2 Duo	1.5GHz (VisualC9.0)	9	9	18	23	43
Intel Core 2 Duo	1.5GHz (VisualC10.0)	9	12	17	24	44
Intel Quad-Core Q6600	2.48GHz (VisualC6.0)	8	13	18	24	47
AMD Athlon X2 4600+	2.4GHz (VisualC6.0)	8	11	16	25	55
Intel Quad-Core Q6600	2.48GHz (OpenWatcom1.8)	8	13	17	26	48
AMD Athlon X2 4600+	2.4GHz (OpenWatcom1.8)	8	11	16	26	58
Intel Pentium M	1.8GHz (VisualC9.0)	8	11	17	27	65
Intel Celeron M	1.0GHz (VisualC10.0)	9	9	19	27	66
Intel Pentium M	1.8GHz (OpenWatcom1.8)	8	10	18	29	67
Intel Celeron M	1.0GHz (VisualC9.0)	9	11	20	27	66
Intel Core 2 Duo	1.5GHz (VisualC6.0)	9	12	23	34	84
Intel Core 2 Duo	1.5GHz (OpenWatcom1.8)	8	13	24	36	83
Intel Pentium 4	2.8GHz (VisualC6.0)	8	12	22	40	90
Intel Celeron M	600MHz (VisualC9.0)	8	14	24	44	108
Intel Pentium 4	2.8GHz (OpenWatcom1.8)	9	13	24	44	101
Intel Celeron M	600MHz (VisualC10.0)	10	14	25	44	108
Intel Celeron M	1.0GHz (VisualC6.0)	8	14	27	46	153
Intel Celeron M	1.0GHz (OpenWatcom1.8)	10	13	28	50	123
Intel Celeron M	600MHz (VisualC6.0)	9	16	40	76	197
Intel Celeron M	600MHz (OpenWatcom1.8)	10	17	46	89	209
Intel Pentium III	400MHz (VisualC10.0)	12	17	50	96	259
Intel Pentium III	400MHz (VisualC9.0)	13	16	51	97	264
Intel Pentium III	400MHz (VisualC6.0)	12	22	70	146	378
Intel Pentium III	400MHz (OpenWatcom1.8)	11	22	84	172	405

```
res(1)
```

```
ans =
```

```

    Name: 'Minimal'
    nBlocks: 3
    BuildTime: 51.2185
    BenchTime: 23.4161
    Tsmin: 4.9727e-06

```

**See Also**

xpctest

**Purpose** Create xPC Target boot disk or DOS Loader files and confirm current environment properties

**Syntax** **MATLAB command line**

```
xpcbootdisk
```

**Description** Function to create an xPC Target boot floppy, CD or DVD boot image, network boot image, or DOS Loader files for the current xPC Target environment. Use the `setxpcenv` function to set environment properties.

- Creating an xPC Target boot floppy consists of writing the bootable kernel image onto the disk. You are asked to insert an empty formatted floppy disk into the 3.5-inch disk drive. At the end, a summary of the creation process is displayed.
- Creating an xPC Target CD/DVD boot image consists of creating the bootable kernel image in a designated area. You can then burn the files to a blank CD/DVD. If you have Microsoft Windows Vista or Microsoft Windows XP Service Pack 2 or 3 with Image Mastering API v2.0 (IMAPIv2.0), `xpcbootdisk` offers to create to the CD or DVD. Otherwise, you must use alternate third-party CD/DVD writing software to write ISO image files.
- Creating an xPC Target network boot image consists of running `xpcnetboot` to start the network boot server process.
- Creating xPC Target DOS Loader files consists of creating the files in a designated area. You can then copy the files to the target computer flash disk.

If you update the environment, you need to update the target boot floppy, CD boot image, network boot image, or DOS Loader files for the new xPC Target environment with the function `xpcbootdisk`.

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

```
xpcbootdisk
```

## See Also

setxpcenv | getxpcenv

## How To

- “Changing Environment Properties with xPC Target Explorer” on page 4-16
- “Changing Environment Properties with a Command-Line Interface for Single Target Computer Systems” on page 4-20

**Purpose** Generate file suitable for use by From File block

**Syntax** `xpcbytes2file(filename, var1, ...,varn)`

## Arguments

<code>filename</code>	Name of the data file from which the From File block distributes data.
<code>var1, ...,varn</code>	Column of data to be output to the model.

## Description

The `xpcbytes2file` function outputs one column of `var1, ..., varn` at every time step. All variables must have the same number of columns; the number of rows and data types can differ.

---

**Note** You might have the data organized such that a row refers to a single time step and not a column. In this case, pass to `xpcbytes2file` the transpose of the variable. To optimize file writes, organize the data in columns.

---

## Examples

In the following example, to use the From File block to output a variable `errorval` (single precision, scalar) and `velocity` (double, width 3) at every time step, you can generate the file with the command:

```
xpcbytes2file('myfile', errorval, velocity)
```

where `errorval` has class `'single'` and dimensions `[1 x N]` and `velocity` has class `'double'` and dimensions `[3 x N]`.

Set up the From File block to output

```
28 bytes  
(1 * sizeof('single') + 3 * sizeof('double'))
```

at every sample time.

# xpcexplr

---

**Purpose** Open xPC Target Explorer

**Syntax** **MATLAB command line**

xpcexplr

**Description** This tool runs on the host computer and allows you to:

- Enter and change xPC Target environment properties
- Create an xPC Target bootable image
- Download, unload, and run target applications
- Monitor signals
- Tune parameters
- Add, remove, and configure xPC Target scopes
- Browse the target file system

---

**Note** xPC Target Explorer runs only on 32-bit host computers. On 64-bit computers, use the MATLAB command-line interface. For further information, see “Configuring Environment From the MATLAB Command Line” on page 4-21.

---

**See Also** setxpcenv | getxpcenv | xpcbootdisk

**How To**

- “Environment Properties for Serial Communication”
- “Environment Properties for Network Communication”

---

<b>Purpose</b>	Compiler settings for xPC Target environment
<b>Syntax</b>	<pre>type = xpcgetCC type = xpcgetCC('Type') [type, location] = xpcgetCC location= xpcgetCC('Location') xpcgetCC('supported') xpcgetCC('installed') [compilers] = xpcgetCC('installed')</pre>
<b>Description</b>	<p><code>type = xpcgetCC</code> and <code>type = xpcgetCC('Type')</code> return the compiler type in <code>type</code>.</p> <p><code>[type, location] = xpcgetCC</code> returns the compiler type and its location in <code>type</code> and <code>location</code>.</p> <p><code>location= xpcgetCC('Location')</code> returns the compiler location in <code>location</code>.</p> <p><code>xpcgetCC('supported')</code> lists supported compiler versions for the xPC Target environment.</p> <p><code>xpcgetCC('installed')</code> lists the xPC Target supported compilers installed on the current host computer</p> <p><code>[compilers] = xpcgetCC('installed')</code> returns the xPC Target supported compilers installed on the current host computer in a structure.</p>
<b>Examples</b>	<p>Return the compiler type.</p> <pre>type = xpcgetCC</pre> <p>Return the compiler type and compiler location.</p> <pre>&gt;&gt; [type, location] = xpcgetCC</pre> <p>Return the xPC Target supported compilers installed on the current host computer in a structure and access the structure fields</p>

# xpcgetCC

---

```
[compilers] = xpcgetCC('installed')
```

```
compilers =
```

```
1x3 struct array with fields:
```

```
    Type
```

```
    Name
```

```
    Location
```

```
compilers.Type
```

```
ans =
```

```
VisualC
```

## See Also

```
xpcsetCC
```



**Purpose** Create kernel to boot target computer over dedicated network

**Syntax** **MATLAB command line**

```
xpcnetboot  
xpcnetboot targetPCname
```

**Arguments**      *targetPCName*      Target computer name as identified in xPC Target Explorer.

**Description**      The xpcnetboot function creates an xPC Target kernel that a target computer within the same network can boot.

This function also starts the following services as server processes:

- Bootstrap protocol (bootp) — xpcbootpserver.exe
- Trivial file transfer protocol (tftp) — xpctftpserver.exe

These processes respond to network boot requests from the target computer.

xpcnetboot creates an xPC Target kernel for the default target computer (as identified in xPC Target Explorer).

xpcnetboot *targetPCname* creates an xPC Target kernel and waits for a request from the target computer named *targetPCname* (as identified in xPC Target Explorer).

**Examples**      In the following example, xpcnetboot creates an xPC Target kernel and waits for a request from the target computer, TargetPC1.

```
xpcnetboot TargetPC1
```

# xpcsetCC

---

**Purpose** Compiler settings for xPC Target environment

**Syntax**

```
xpcsetCC('setup')
xpcsetCC('location')
xpcsetCC('type')
xpcsetCC(type, location)
```

**Description** `xpcsetCC('setup')` queries the host computer for installed C compilers that the xPC Target environment supports. You can then select the C compiler. See “Configuring the xPC Target Host Computer for Your C Compiler with a Command-Line Interface” for an example of how to use this function.

`xpcsetCC('location')` sets the compiler location.

`xpcsetCC('type')` sets the compiler type. 'type' must be one of the following:

- VISUALC  
Microsoft Visual Studio C compiler
- WATCOM  
Open Watcom C compiler

---

**Note** The WATCOM 1.8 compiler will not compile all models. If a compilation fails, use a Microsoft compiler instead. WATCOM compiler support will be removed in a future release.

---

`xpcsetCC(type, location)` sets the compiler type and location.

**See Also** `xpcgetCC`

**How To**

- “Configuring the xPC Target Host Computer for Your C Compiler with a Command-Line Interface”

**Purpose** Package for all xPC Target MATLAB classes

**Description** Use xpctarget package objects to access all of the MATLAB command line capabilities.

### Functions

Assign these object creation functions to a MATLAB variable to get access to the properties and methods of the class.

Function	Description
xpctarget.fs	Create file system object
xpctarget.ftp	Create file transfer protocol (FTP) object
xpctarget.targets	Create container object to manage target computer environment collection objects
xpctarget.xpc	Create target object representing target application

# xpctarget.env Class

---

**Purpose** Stores target environment properties

**Description** Each `xpctarget.env` Class object contains the environment properties for a particular target computer. A collection of these objects is stored in an `xpctarget.targets` Class object. An individual object in a collection is accessed via the `xpctarget.targets.Item (env collection object)` method.

## Methods

Method	Description
<code>xpctarget.env.get (env object)</code>	Return property values for an environment object
<code>xpctarget.env.set (env object)</code>	Change property values for an environment object

## Properties

Environment Property	Description	Writable
Name	Target computer name.	Yes
HostTargetComm	<p>Values are 'RS232' and 'TcpIp'.</p> <p>From the xPC Target Explorer window <b>Host target communication</b> list, select either RS232 or TCP/IP.</p> <p>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.</p> <hr/> <p><b>Note</b> RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.</p> <hr/>	Yes
TargetRAMSizeMB	<p>Values are 'Auto' and 'Manual'.</p> <p>From the xPC Target Explorer window <b>Target RAM size</b> list, select either Auto or Manual. If you select Manual, enter the amount of RAM, in megabytes, installed on the target computer. This property is set by default to Auto.</p> <p><b>Target RAM size</b> defines the total amount of installed RAM in the target computer. This RAM is used for the kernel, target application, data logging, and other functions that use the heap.</p> <p>If <b>Target RAM size</b> is set to Auto, the target application automatically determines the amount of memory up to 64 MB. If the</p>	Yes

# xpctarget.env Class

Environment Property	Description	Writable
	<p>target computer does not contain more than 64 MB of RAM, or you do not want to use more than 64 MB, select <code>Auto</code>. If the target computer has more than 64 MB of RAM, and you want to use more than 64 MB, select <code>Manual</code>, and enter the amount of RAM installed in the target computer.</p> <hr/> <p><b>Note</b> The xPC Target kernel can use only 2 GB of memory.</p> <hr/>	
<p>MaxModelSize</p>	<p>BootFloppy and DOSLoader modes ignore this value.</p> <p>Values are '1MB', '4MB', and '16MB'.</p> <p>From the xPC Target Explorer window <b>Maximum model size</b> list, select either 1 MB, 4 MB, or 16 MB. This value is unavailable for BootFloppy or DOSLoader modes.</p> <p>Choosing the maximum model size reserves the specified amount of memory on the target computer for the target application. The remaining memory is used by the kernel and by the heap for data logging.</p> <p>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.</p> <p>Note that you cannot build a 16 MB target application to run in StandAlone mode.</p>	<p>Yes</p>

Environment Property	Description	Writable
TargetScope	<p>Values are 'Disabled' and 'Enabled'.</p> <p>From the xPC Target Explorer window <b>Enable target scope</b> list, select either Enabled or Disabled.</p> <p>The property TargetScope is set by default to Enabled. If you set TargetScope to Disabled, the target computer displays information as text.</p> <p>To use all the features of the target scope, you also need to install a keyboard on the target computer.</p>	Yes
TargetBoot	<p>Values are 'BootFloppy', 'CDBoot', 'DOSLoader', 'NetworkBoot', and 'StandAlone'.</p> <p>From the xPC Target Explorer window target computer configuration pane, select one of the following tabs: <b>Boot Floppy</b>, <b>CD Boot</b>, <b>DOS Loader</b>, <b>Network Boot</b>, or <b>Standalone</b>.</p> <p>If your license file does not include the license for the xPC Target Embedded Option product, your only options are BootFloppy, CDBoot, DOSLoader, and NetworkBoot. With the xPC Target Embedded Option product licensed and installed, you have the additional choice of Standalone.</p>	Yes
BootFloppyLocation	Drive name for creation of 3.5-inch target boot disk.	Yes
DOSLoaderLocation	Location of DOSLoader files to boot target computers from devices other than 3.5-inch disk or CD.	Yes

## xpctarget.env Class

Environment Property	Description	Writable
CDBootImageLocation	Location of cdboot.iso file for creation of CD target boot disk.	Yes
EmbeddedOption	Values are 'Disabled' and 'Enabled'. This property is read only.  Note that the xPC Target Embedded Option product is enabled only if you purchase an additional license.	Yes
SecondaryIDE	Values are 'off' and 'on'. Set this value to 'on' only if you want to use the disks connected to a secondary IDE controller. If you do not have disks connected to the secondary IDE controller, leave this value set to 'off'.	Yes
RS232HostPort	Values are 'COM1' and 'COM2'.  From the xPC Target Explorer window <b>Host port</b> list, select either COM1 or COM2 for the connection on the host computer. The xPC Target software automatically determines the COM port on the target computer.  Before you can select an RS-232 port, you need to set the HostTargetComm property to RS232.	Yes
RS232Baudrate	Values are '115200', '57600', '38400', '19200', '9600', '4800', '2400', and '1200'.  From the <b>Baud rate</b> list, select 115200, 57600, 38400, 19200, 9600, 4800, 2400, or 1200.	Yes



Environment Property	Description	Writable
TcpIpTargetAddress	<p>Value is 'xxx.xxx.xxx.xxx'.</p> <p>In the xPC Target Explorer window <b>Target PC IP address</b> box, enter a valid IP address for your target computer. Ask your system administrator for this value.</p> <p>For example, 192.168.0.10.</p>	Yes
TcpIpTargetPort	<p>Value is 'xxxxx'.</p> <p>In the xPC Target Explorer window <b>TcpIp target port</b> box, enter a value greater than 20000.</p> <p>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 computer.</p>	Yes
TcpIpSubNetMask	<p>Value is 'xxx.xxx.xxx.xxx'.</p> <p>In the xPC Target Explorer window <b>LAN subnet mask address</b> text box, enter the subnet mask of your LAN. Ask your system administrator for this value.</p> <p>For example, your subnet mask could be 255.255.255.0.</p>	Yes

# xpctarget.env Class

Environment Property	Description	Writable
TcpIpGateway	<p>Value is 'xxx.xxx.xxx.xxx'.</p> <p>In the xPC Target Explorer window <b>TcpIp gateway address</b> 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 computer.</p> <p>If you communicate with your target computer 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.</p>	Yes
TcpIpTargetDriver	<p>Values are 'NE2000', 'SMC91C9X', 'I82559', 'RTLANCE', 'R8139', '3C90x', and 'NS83815'.</p> <p>From the xPC Target Explorer window <b>TcpIp target driver</b> list, select NE2000, SMC91C9X, I82559, RTLANCE, R8139, 3C90x, or NS83815.</p>	Yes

Environment Property	Description	Writable
TcpIpTargetBusType	<p>Values are 'PCI' and 'ISA'.</p> <p>From the xPC Target Explorer window <b>TcpIp target bus type</b> list, select either PCI or ISA. This property is set by default to PCI, and determines the bus type of your target computer. You do not need to define a bus type for your host computer, which can be the same or different from the bus type in your target computer.</p> <p>If TcpIpTargetBusType is set to PCI, then the properties TcpIpISAMemPort and TcpIpISAIRQ have no effect on TCP/IP communication.</p> <p>If you are using an ISA bus card, set TcpIpTargetBusType to ISA and enter values for TcpIpISAMemPort and TcpIpISAIRQ.</p>	Yes
TcpIpTargetISAIRQ	<p>Value is 'n', where <i>n</i> is between 4 and 15.</p> <p>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.</p> <p>On your ISA bus card, assign an IRQ and I/O-port base address by moving the jumpers on the card.</p> <p>Set the IRQ to 5, 10, or 11. If one of these hardware settings leads to a conflict in your target computer, choose another IRQ and</p>	Yes

## xpctarget.env Class

Environment Property	Description	Writable
	make the corresponding changes to your jumper settings.	
TcpIpTargetISAMem Port	<p>Value is '0xnnnn'.</p> <p>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.</p> <p>On your ISA bus card, assign an IRQ and I/O port base address by moving the jumpers on the card.</p> <p>Set the I/O port base address to around 0x300. If one of these hardware settings leads to a conflict in your target computer, choose another I/O port base address and make the corresponding changes to your jumper settings.</p>	Yes
TargetMACAddress	Physical target computer MAC address from which to accept boot requests when booting within a dedicated network. Enter the address as six pairs of hexadecimal numbers, separated by colons (:).	Yes

Environment Property	Description	Writable
CCompiler	<hr/> <b>Note</b> <ul style="list-style-type: none"><li>• CCompiler and CompilerPath will be removed in a future version. Use the xpcgetCC function to configure the compiler instead.</li><li>• WATCOM compiler support will be removed in a future release. Use a Microsoft compiler instead.</li></ul> <hr/> Values are 'Watcom' and 'VisualC'. From the xPC Target Explorer window compiler list, select either Watcom or VisualC.	Yes
CompilerPath	Value is a valid compiler root folder. Enter the path where you installed a Watcom C/C++ or Microsoft Visual Studio C/C++ compiler.  If the path is invalid or the folder does not contain the compiler, an error message appears when you build a target application.	Yes

# xpctarget.env.get (env object)

---

**Purpose** Return target environment property values

**Syntax** MATLAB command line

```
get(env_object)
get(env_object, 'property_name1', 'property_value1',
'property_name2', 'property_value2', . . .)
env_object.get('property_name1', 'property_value1')
get(env_object, property_name_vector, property_value_vector)
env_object.property_name = property_value
```

**Arguments**

<code>env_object</code>	Name of a target environment object.
<code>'property_name'</code>	Name of a target environment object property. Always use quotation marks.
<code>property_value</code>	Value for a target environment object property. Always use quotation marks for character strings; quotation marks are optional for numbers.
<code>parameter_name</code>	The letter p followed by the parameter index. For example, p0, p1, p2.

**Description**

Not all properties are user writable. Get an individual environment object via the `xpctarget.targets.Item` (env collection object) method.

The environment properties for a target environment 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:

## xpctarget.env.get (env object)

Environment Property	Description	Writable
Name	Target computer name.	Yes
HostTargetComm	<p>Values are 'RS232' and 'TcpIp'.</p> <p>From the xPC Target Explorer window <b>Host target communication</b> list, select either RS232 or TCP/IP.</p> <p>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.</p> <hr/> <p><b>Note</b> RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.</p> <hr/>	Yes
TargetRAMSizeMB	<p>Values are 'Auto' and 'Manual'.</p> <p>From the xPC Target Explorer window <b>Target RAM size</b> list, select either Auto or Manual. If you select Manual, enter the amount of RAM, in megabytes, installed on the target computer. This property is set by default to Auto.</p> <p><b>Target RAM size</b> defines the total amount of installed RAM in the target computer. This RAM is used for the kernel, target application, data logging, and other functions that use the heap.</p> <p>If <b>Target RAM size</b> is set to Auto, the target application automatically determines the amount of memory up to 64 MB. If the target computer does not contain more than 64 MB of RAM, or you do not want to use</p>	Yes

## xpctarget.env.get (env object)

Environment Property	Description	Writable
	<p>more than 64 MB, select Auto. If the target computer 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 computer.</p> <hr/> <p><b>Note</b> The xPC Target kernel can use only 2 GB of memory.</p> <hr/>	
MaxModelSize	<p>BootFloppy and DOSLoader modes ignore this value.</p> <p>Values are '1MB', '4MB', and '16MB'.</p> <p>From the xPC Target Explorer window <b>Maximum model size</b> list, select either 1 MB, 4 MB, or 16 MB. This value is unavailable for BootFloppy or DOSLoader modes.</p> <p>Choosing the maximum model size reserves the specified amount of memory on the target computer for the target application. The remaining memory is used by the kernel and by the heap for data logging.</p> <p>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.</p> <p>Note that you cannot build a 16 MB target application to run in StandAlone mode.</p>	Yes



## xpctarget.env.get (env object)

Environment Property	Description	Writable
TargetScope	<p>Values are 'Disabled' and 'Enabled'.</p> <p>From the xPC Target Explorer window <b>Enable target scope</b> list, select either Enabled or Disabled.</p> <p>The property TargetScope is set by default to Enabled. If you set TargetScope to Disabled, the target computer displays information as text.</p> <p>To use all the features of the target scope, you also need to install a keyboard on the target computer.</p>	Yes
TargetBoot	<p>Values are 'BootFloppy', 'CDBoot', 'DOSLoader', 'NetworkBoot', and 'StandAlone'.</p> <p>From the xPC Target Explorer window target computer configuration pane, select one of the following tabs: <b>Boot Floppy</b>, <b>CD Boot</b>, <b>DOS Loader</b>, <b>Network Boot</b>, or <b>Standalone</b>.</p> <p>If your license file does not include the license for the xPC Target Embedded Option product, your only options are BootFloppy, CDBoot, DOSLoader, and NetworkBoot. With the xPC Target Embedded Option product licensed and installed, you have the additional choice of Standalone.</p>	Yes
BootFloppyLocation	Drive name for creation of 3.5-inch target boot disk.	Yes
DOSLoaderLocation	Location of DOSLoader files to boot target computers from devices other than 3.5-inch disk or CD.	Yes

## xpctarget.env.get (env object)

Environment Property	Description	Writable
CDBootImageLocation	Location of cdboot.iso file for creation of CD target boot disk.	Yes
EmbeddedOption	Values are 'Disabled' and 'Enabled'. This property is read only.  Note that the xPC Target Embedded Option product is enabled only if you purchase an additional license.	Yes
SecondaryIDE	Values are 'off' and 'on'. Set this value to 'on' only if you want to use the disks connected to a secondary IDE controller. If you do not have disks connected to the secondary IDE controller, leave this value set to 'off'.	Yes
RS232HostPort	Values are 'COM1' and 'COM2'.  From the xPC Target Explorer window <b>Host port</b> list, select either COM1 or COM2 for the connection on the host computer. The xPC Target software automatically determines the COM port on the target computer.  Before you can select an RS-232 port, you need to set the HostTargetComm property to RS232.	Yes
RS232Baudrate	Values are '115200', '57600', '38400', '19200', '9600', '4800', '2400', and '1200'.  From the <b>Baud rate</b> list, select 115200, 57600, 38400, 19200, 9600, 4800, 2400, or 1200.	Yes

## xpctarget.env.get (env object)

Environment Property	Description	Writable
TcpIpTargetAddress	<p>Value is 'xxx.xxx.xxx.xxx'.</p> <p>In the xPC Target Explorer window <b>Target PC IP address</b> box, enter a valid IP address for your target computer. Ask your system administrator for this value.</p> <p>For example, 192.168.0.10.</p>	Yes
TcpIpTargetPort	<p>Value is 'xxxxx'.</p> <p>In the xPC Target Explorer window <b>TcpIp target port</b> box, enter a value greater than 20000.</p> <p>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 computer.</p>	Yes
TcpIpSubNetMask	<p>Value is 'xxx.xxx.xxx.xxx'.</p> <p>In the xPC Target Explorer window <b>LAN subnet mask address</b> text box, enter the subnet mask of your LAN. Ask your system administrator for this value.</p> <p>For example, your subnet mask could be 255.255.255.0.</p>	Yes

## xpctarget.env.get (env object)

Environment Property	Description	Writable
TcpIpGateway	<p>Value is 'xxx.xxx.xxx.xxx'.</p> <p>In the xPC Target Explorer window <b>TcpIp gateway address</b> 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 computer.</p> <p>If you communicate with your target computer 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.</p>	Yes
TcpIpTargetDriver	<p>Values are 'NE2000', 'SMC91C9X', 'I82559', 'RTLANCE', 'R8139', '3C90x', and 'NS83815'.</p> <p>From the xPC Target Explorer window <b>TcpIp target driver</b> list, select NE2000, SMC91C9X, I82559, RTLANCE, R8139, 3C90x, or NS83815.</p>	Yes

Environment Property	Description	Writable
TcpIpTargetBusType	<p>Values are 'PCI' and 'ISA'.</p> <p>From the xPC Target Explorer window <b>TcpIp target bus type</b> list, select either PCI or ISA. This property is set by default to PCI, and determines the bus type of your target computer. You do not need to define a bus type for your host computer, which can be the same or different from the bus type in your target computer.</p> <p>If TcpIpTargetBusType is set to PCI, then the properties TcpIpISAMemPort and TcpIpISAIRQ have no effect on TCP/IP communication.</p> <p>If you are using an ISA bus card, set TcpIpTargetBusType to ISA and enter values for TcpIpISAMemPort and TcpIpISAIRQ.</p>	Yes
TcpIpTargetISAIRQ	<p>Value is 'n', where <i>n</i> is between 4 and 15.</p> <p>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.</p> <p>On your ISA bus card, assign an IRQ and I/O-port base address by moving the jumpers on the card.</p> <p>Set the IRQ to 5, 10, or 11. If one of these hardware settings leads to a conflict in your target computer, choose another IRQ and</p>	Yes

## xpctarget.env.get (env object)

Environment Property	Description	Writable
	make the corresponding changes to your jumper settings.	
TcpIpTargetISAMem Port	<p>Value is '0xnnnn'.</p> <p>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.</p> <p>On your ISA bus card, assign an IRQ and I/O port base address by moving the jumpers on the card.</p> <p>Set the I/O port base address to around 0x300. If one of these hardware settings leads to a conflict in your target computer, choose another I/O port base address and make the corresponding changes to your jumper settings.</p>	Yes
TargetMACAddress	Physical target computer MAC address from which to accept boot requests when booting within a dedicated network. Enter the address as six pairs of hexadecimal numbers, separated by colons (:).	Yes

## xpctarget.env.get (env object)

Environment Property	Description	Writable
CCompiler	<p><b>Note</b></p> <ul style="list-style-type: none"><li>• CCompiler and CompilerPath will be removed in a future version. Use the xpcgetCC function to configure the compiler instead.</li><li>• WATCOM compiler support will be removed in a future release. Use a Microsoft compiler instead.</li></ul> <hr/> <p>Values are 'Watcom' and 'VisualC'. From the xPC Target Explorer window compiler list, select either Watcom or VisualC.</p>	Yes
CompilerPath	<p>Value is a valid compiler root folder. Enter the path where you installed a Watcom C/C++ or Microsoft Visual Studio C/C++ compiler.</p> <p>If the path is invalid or the folder does not contain the compiler, an error message appears when you build a target application.</p>	Yes

### See Also

xpctarget.env.set (env object)

# xpctarget.env.set (env object)

---

**Purpose** Change target environment object property values

**Syntax** MATLAB command line

```
set(env_object)
set(env_object, 'property_name1', 'property_value1',
'property_name2', 'property_value2', . . .)
env_object.set('property_name1', 'property_value1')
set(env_object, property_name_vector,
property_value_vector)
env_object.property_name = property_value
```

## Arguments

env_object	Name of a target environment object.
'property_name'	Name of a target environment object property. Always use quotation marks.
property_value	Value for a target environment object property. Always use quotation marks for character strings; quotation marks are optional for numbers.

## Description

Not all properties are user writable. Get an individual environment object via the `xpctarget.targets.Item (env collection object)` method.

Properties must be entered in pairs or, using the alternate syntax, as one-dimensional cell arrays of the same size. This means they must 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 writable properties for a target environment object are listed in the following table. This table includes a description of the properties:



## xpctarget.env.set (env object)

Environment Property	Description	Writable
Name	Target computer name.	Yes
HostTargetComm	<p>Values are 'RS232' and 'TcpIp'.</p> <p>From the xPC Target Explorer window <b>Host target communication</b> list, select either RS232 or TCP/IP.</p> <p>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.</p> <hr/> <p><b>Note</b> RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.</p> <hr/>	Yes
TargetRAMSizeMB	<p>Values are 'Auto' and 'Manual'.</p> <p>From the xPC Target Explorer window <b>Target RAM size</b> list, select either Auto or Manual. If you select Manual, enter the amount of RAM, in megabytes, installed on the target computer. This property is set by default to Auto.</p> <p><b>Target RAM size</b> defines the total amount of installed RAM in the target computer. This RAM is used for the kernel, target application, data logging, and other functions that use the heap.</p>	Yes

## xpctarget.env.set (env object)

Environment Property	Description	Writable
	<p>If <b>Target RAM size</b> is set to Auto, the target application automatically determines the amount of memory up to 64 MB. If the target computer does not contain more than 64 MB of RAM, or you do not want to use more than 64 MB, select Auto. If the target computer 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 computer.</p> <hr/> <p><b>Note</b> The xPC Target kernel can use only 2 GB of memory.</p> <hr/>	
MaxModelSize	<p>BootFloppy and DOSLoader modes ignore this value.</p> <p>Values are '1MB', '4MB', and '16MB.</p> <p>From the xPC Target Explorer window <b>Maximum model size</b> list, select either 1 MB, 4 MB, or 16 MB. This value is unavailable for BootFloppy or DOSLoader modes.</p> <p>Choosing the maximum model size reserves the specified amount of memory on the target computer for the target application. The remaining memory is used by the kernel and by the heap for data logging.</p>	Yes

## xpctarget.env.set (env object)

Environment Property	Description	Writable
	<p>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.</p> <p>Note that you cannot build a 16 MB target application to run in StandAlone mode.</p>	
TargetScope	<p>Values are 'Disabled' and 'Enabled'.</p> <p>From the xPC Target Explorer window <b>Enable target scope</b> list, select either Enabled or Disabled.</p> <p>The property TargetScope is set by default to Enabled. If you set TargetScope to Disabled, the target computer displays information as text.</p> <p>To use all the features of the target scope, you also need to install a keyboard on the target computer.</p>	Yes
DOSLoaderLocation	Location of DOSLoader files to boot target computers from devices other than 3.5-inch disk or CD.	Yes
BootFloppyLocation	Drive name for creation of 3.5-inch target boot disk.	Yes
CDBootImageLocation	Location of cdboot.iso file for creation of CD target boot disk.	Yes

## xpctarget.env.set (env object)

Environment Property	Description	Writable
TargetBoot	<p>Values are 'BootFloppy', 'CDBoot', 'DOSLoader', 'NetworkBoot', and 'StandAlone'.</p> <p>From the xPC Target Explorer window target computer configuration pane, select one of the following tabs: <b>Boot Floppy</b>, <b>CD Boot</b>, <b>DOS Loader</b>, <b>Network Boot</b>, or <b>Standalone</b>.</p> <p>If your license file does not include the license for the xPC Target Embedded Option product, your only options are BootFloppy, CDBoot, DOSLoader, and NetworkBoot. With the xPC Target Embedded Option product licensed and installed, you have the additional choice of Standalone.</p>	Yes
EmbeddedOption	<p>Values are 'Disabled' and 'Enabled'. This property is read only.</p> <p>Note that the xPC Target Embedded Option product is enabled only if you purchase an additional license.</p>	Yes
SecondaryIDE	<p>Values are 'off' and 'on'. Set this value to 'on' only if you want to use the disks connected to a secondary IDE controller. If you do not have disks connected to the secondary IDE controller, leave this value set to 'off'.</p>	Yes

## xpctarget.env.set (env object)

Environment Property	Description	Writable
RS232HostPort	<p>Values are 'COM1' and 'COM2'.</p> <p>From the xPC Target Explorer window <b>Host port</b> list, select either COM1 or COM2 for the connection on the host computer. The xPC Target software automatically determines the COM port on the target computer.</p> <p>Before you can select an RS-232 port, you need to set the HostTargetComm property to RS232.</p>	Yes
RS232Baudrate	<p>Values are '115200', '57600', '38400', '19200', '9600', '4800', '2400', and '1200'.</p> <p>From the <b>Baud rate</b> list, select 115200, 57600, 38400, 19200, 9600, 4800, 2400, or 1200.</p>	Yes
TcpIpTargetAddress	<p>Value is 'xxx.xxx.xxx.xxx'.</p> <p>In the xPC Target Explorer window <b>Target PC IP address</b> box, enter a valid IP address for your target computer. Ask your system administrator for this value.</p> <p>For example, 192.168.0.10.</p>	Yes

## xpctarget.env.set (env object)

Environment Property	Description	Writable
TcpIpTargetPort	<p>Value is 'xxxxx'.</p> <p>In the xPC Target Explorer window <b>TcpIp target port</b> box, enter a value greater than 20000.</p> <p>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 computer.</p>	Yes
TcpIpSubNetMask	<p>Value is 'xxx.xxx.xxx.xxx'.</p> <p>In the xPC Target Explorer window <b>LAN subnet mask address</b> text box, enter the subnet mask of your LAN. Ask your system administrator for this value.</p> <p>For example, your subnet mask could be 255.255.255.0.</p>	Yes
TcpIpGateway	<p>Value is 'xxx.xxx.xxx.xxx'.</p> <p>In the xPC Target Explorer window <b>TcpIp gateway address</b> 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 computer.</p> <p>If you communicate with your target computer from within a LAN that uses gateways, and your host and target computers are connected through a gateway, then you need to enter a</p>	Yes

## xpctarget.env.set (env object)

Environment Property	Description	Writable
	value for this property. If your LAN does not use gateways, you do not need to change this property. Ask your system administrator.	
TcpIpTargetDriver	<p>Values are 'NE2000', 'SMC91C9X', 'I82559', 'RTLANCE', 'R8139', '3C90x', and 'NS83815'.</p> <p>From the xPC Target Explorer window <b>TcpIp target driver</b> list, select NE2000, SMC91C9X, I82559, RTLANCE, 'R8139', 3C90x, and NS83815.</p>	Yes
TcpIpTargetBusType	<p>Values are 'PCI' and 'ISA'.</p> <p>From the xPC Target Explorer window <b>TcpIp target bus type</b> list, select either PCI or ISA. This property is set by default to PCI, and determines the bus type of your target computer. You do not need to define a bus type for your host computer, which can be the same or different from the bus type in your target computer.</p> <p>If TcpIpTargetBusType is set to PCI, then the properties TcpIpISAMemPort and TcpIpISAIRQ have no effect on TCP/IP communication.</p> <p>If you are using an ISA bus card, set TcpIpTargetBusType to ISA and enter values for TcpIpISAMemPort and TcpIpISAIRQ.</p>	Yes

## xpctarget.env.set (env object)

Environment Property	Description	Writable
TcpIpTargetISAMem Port	<p>Value is '0xnxxx'.</p> <p>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.</p> <p>On your ISA bus card, assign an IRQ and I/O port base address by moving the jumpers on the card.</p> <p>Set the I/O port base address to around 0x300. If one of these hardware settings leads to a conflict in your target computer, choose another I/O port base address and make the corresponding changes to your jumper settings.</p>	Yes
TargetMACAddress	<p>Physical target computer MAC address from which to accept boot requests when booting within a dedicated network. Enter the address as six pairs of hexadecimal numbers, separated by colons (:).</p>	Yes



Environment Property	Description	Writable
CCompiler	<hr/> <p><b>Note</b></p> <ul style="list-style-type: none"> <li>• CCompiler and CompilerPath will be removed in a future version. Use the xpcsetCC function to configure the compiler instead.</li> <li>• WATCOM compiler support will be removed in a future release. Use a Microsoft compiler instead.</li> </ul> <hr/> <p>Values are 'Watcom' and 'VisualC'. From the xPC Target Explorer window compiler list, select either Watcom or VisualC.</p>	Yes
CompilerPath	<p>Value is a valid compiler root folder. Enter the path where you installed a Watcom C/C++ or Microsoft Visual Studio C/C++ compiler.</p> <p>If the path is invalid or the folder does not contain the compiler, an error message appears when you build a target application.</p>	Yes
TcpIpTargetISAIRQ	<p>Value is 'n', where <i>n</i> is between 4 and 15.</p> <p>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</p>	Yes

## xpctarget.env.set (env object)

---

Environment Property	Description	Writable
	<p>jumper settings or ROM settings on the ISA-bus Ethernet card.</p> <p>On your ISA bus card, assign an IRQ and I/O-port base address by moving the jumpers on the card.</p> <p>Set the IRQ to 5, 10, or 11. If one of these hardware settings leads to a conflict in your target computer, choose another IRQ and make the corresponding changes to your jumper settings.</p>	

### See Also

`xpctarget.env.get (env object)`

**Purpose** Manage the directories and files on the target computer

**Description** This class includes the directory methods from `xpctarget.fsbase` Class and implements file access methods used on the target computer.

## Constructor

Constructor	Description
<code>xpctarget.fs</code>	Create file system object

## Methods

These methods are inherited from `xpctarget.fsbase` Class.

Method	Description
<code>xpctarget.fsbase.cd</code>	Change folder on target computer
<code>xpctarget.fsbase.dir</code>	List contents of current folder on target computer
<code>xpctarget.fsbase.mkdir</code>	Make folder on target computer
<code>xpctarget.fsbase.pwd</code>	Current folder path of target computer
<code>xpctarget.fsbase.rmdir</code>	Remove folder from target computer

These methods are specific to class `fs`.

Method	Description
<code>xpctarget.fs.diskinfo</code>	Information about target computer drive
<code>xpctarget.fs.fclose</code>	Close open target computer file(s)
<code>xpctarget.fs.fileinfo</code>	Target computer file information
<code>xpctarget.fs.filetable</code>	Information about open files in target computer file system
<code>xpctarget.fs.fopen</code>	Open target computer file for reading

## xpctarget.fs Class

---

Method	Description
<code>xpctarget.fs.fread</code>	Read open target computer file
<code>xpctarget.fs.fwrite</code>	Write binary data to open target computer file
<code>xpctarget.fs.getfilesize</code>	Size of file on target computer
<code>xpctarget.fs.removefile</code>	Remove file from target computer

**Purpose** Create xPC Target file system object

**Syntax** MATLAB command line

```
fileSYS_object = xpctarget.fs('mode', 'arg1', 'arg2')
```

**Arguments**

`fileSYS_object` Variable name to reference the file system object.  
`mode` Optionally, enter the communication mode:

---

**Note** RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.

---

TCPIP Specify TCP/IP connection with target computer.

RS232 Specify RS-232 connection with target computer.

`arg1` Optionally, enter an argument based on the mode value:

IP address If mode is 'TCPIP', enter the IP address of the target computer.

COM port If mode is 'RS232', enter the host COM port.

`arg2` Optionally, enter an argument based on the mode value:

Port If mode is 'TCPIP', enter the port number for the target computer.

Baud rate If mode is 'RS232', enter the baud rate for the connection between the host and target computer.

# xpctarget.fs

---

## Description

Constructor of a file system object (`xpctarget.fs` Class). The file system object represents the file system on the target computer. You work with the file system by changing the file system object using methods.

If you have one target computer object, or if you designate a target computer as the default one in your system, use the syntax

```
fileSYS_object=xpctarget.fs
```

If you have multiple target computers in your system, or if you want to identify a target computer with the file system object, use the following syntax to create the additional file system objects.

```
fileSYS_object=xpctarget.fs('mode', 'arg1', 'arg2')
```

## Examples

In the following example, a file system object for a target computer with an RS-232 connection is created.

```
fs1=xpctarget.fs('RS232', 'COM1', '115200')
```

```
fs1 =  
xpctarget.fs
```

Optionally, if you have an `xpctarget.xpc` object, you can construct an `xpctarget.fs` object by passing the `xpctarget.xpc` object variable to the `xpctarget.fs` constructor as an argument.

```
>> tg1=xpctarget.xpc('RS232', 'COM1', '115200');  
>> fs2=xpctarget.fs(tg1)
```

```
fs2 =  
  
xpctarget.fs
```

**Purpose** Information about target computer drive

**Syntax** MATLAB command line

```
diskinfo(filesys_obj,target_PC_drive)
filesys_obj.diskinfo(target_PC_drive)
```

**Arguments**

<code>filesys_obj</code>	Name of the <code>xpctarget.fs</code> file system object.
<code>target_PC_drive</code>	Name of the target computer drive for which to return information.

**Description** Method of `xpctarget.fs` objects. From the host computer, returns disk information for the specified target computer drive.

# xpctarget.fs.diskinfo

---

## Examples

Return disk information for the target computer C:\ drive for the file system object fsys.

```
diskinfo(fsys, 'C:\') or fsys.diskinfo('C:\')  
ans =
```

```
          Label: 'SYSTEM '  
    DriveLetter: 'C'  
      Reserved: ''  
  SerialNumber: 1.0294e+009  
FirstPhysicalSector: 63  
          FATType: 32  
          FATCount: 2  
    MaxDirEntries: 0  
  BytesPerSector: 512  
SectorsPerCluster: 4  
    TotalClusters: 2040293  
    BadClusters: 0  
    FreeClusters: 1007937  
          Files: 19968  
    FileChains: 22480  
    FreeChains: 1300  
LargestFreeChain: 64349
```



<b>Purpose</b>	Close open target computer file(s)				
<b>Syntax</b>	<b>MATLAB command line</b>  fclose(filesys_obj,file_ID) filesys_obj.fclose(file_ID)				
<b>Arguments</b>	<table><tr><td>filesys_obj</td><td>Name of the xpctarget.fs file system object.</td></tr><tr><td>file_ID</td><td>File identifier of the file to close.</td></tr></table>	filesys_obj	Name of the xpctarget.fs file system object.	file_ID	File identifier of the file to close.
filesys_obj	Name of the xpctarget.fs file system object.				
file_ID	File identifier of the file to close.				
<b>Description</b>	Method of xpctarget.fs objects. From the host computer, closes one or more open files in the target computer file system (except standard input, output, and error). The file_ID argument is the file identifier associated with an open file (see xpctarget.fs.fopen and xpctarget.fs.filetable). You cannot have more than eight files open in the file system.				
<b>Examples</b>	Close the open file identified by the file identifier h in the file system object fsys.  fclose(fsys,h) or fsys.fclose(h)				
<b>See Also</b>	fclose   xpctarget.fs.fopen   xpctarget.fs.fread   xpctarget.fs.filetable   xpctarget.fs.fwrite				

# xpctarget.fs.fileinfo

---

**Purpose** Target computer file information

**Syntax** MATLAB command line

```
fileinfo(filesys_obj,file_ID)
filesys_obj.fileinfo(file_ID)
```

**Arguments**

filesys_obj	Name of the xpctarget.fs file system object.
file_ID	File identifier of the file for which to get file information.

**Description** Method of xpctarget.fs objects. From the host computer, gets the information for the file associated with file\_ID.

**Examples** Return file information for the file associated with the file identifier h in the file system object fsys.

```
fileinfo(fsys,h) or fsys.fileinfo(h)
ans =
        FilePos: 0
        AllocatedSize: 12288
        ClusterChains: 1
        VolumeSerialNumber: 1.0450e+009
        FullName: 'C:\DATA.DAT'
```

**Purpose** Information about open files in target computer file system

**Syntax** MATLAB command line

```
filetable(filesys_obj)  
filesys_obj.filetable
```

**Arguments** filesys\_obj Name of the xpctarget.fs file system object.

**Description** Method of xpctarget.fs objects. From the host computer, displays a table of the open files in the target computer file system. You cannot have more than eight files open in the file system.

**Examples** Return a table of the open files in the target computer file system for the file system object fsys.

```
filetable(fsys) or fsys.filetable
```

```
ans =
```

Index	Handle	Flags	FilePos	Name
0	00060000	R__	8512	C:\DATA.DAT
1	00080001	R__	0	C:\DATA1.DAT
2	000A0002	R__	8512	C:\DATA2.DAT
3	000C0003	R__	8512	C:\DATA3.DAT
4	001E000S	R__	0	C:\DATA4.DAT

The table returns the open file handles in hexadecimal. To convert a handle to one that other xpctarget.fs methods, such as fclose, can use, use the hex2dec function.

```
h1 = hex2dec('001E0001')
```

```
h1 =
```

```
1966081
```

To close that file, use the xpctarget.fs fclose method.

# xpctarget.fs.filetable

---

```
fsys.fclose(h1);
```

## **See Also**

`xpctarget.fs.fopen` | `xpctarget.fs.fclose`

**Purpose** Open target computer file for reading

**Syntax** MATLAB command line

```
file_ID = fopen(file_obj, 'file_name')  
file_ID = file_obj.fopen('file_name')  
file_ID = fopen(file_obj, 'file_name', permission)  
file_ID = file_obj.fopen('file_name', permission)
```

**Arguments**

file_obj	Name of the xpctarget.fs object.
'file_name'	Name of the target computer to open.
permission	Values are 'r', 'w', 'a', 'r+', 'w+', or 'a+'. This argument is optional with 'r' as the default value.

**Description** Method of xpctarget.fs objects. From the host computer, opens the specified filename on the target computer for binary access.

The permission argument values are

- 'r'  
Open the file for reading (default). The method does nothing if the file does not already exist.
- 'w'  
Open the file for writing. The method creates the file if it does not already exist.
- 'a'  
Open the file for appending to the file. Initially, the file pointer is at the end of the file. The method creates the file if it does not already exist.
- 'r+'

# xpctarget.fs.fopen

---

Open the file for reading and writing. Initially, the file pointer is at the beginning of the file. The method does nothing if the file does not already exist.

- 'w+'

Open the file for reading and writing. The method empties the file first, if the file already exists and has content, and places the file pointer at the beginning of the file. The method creates the file if it does not already exist.

- 'a+'

Open the file for reading and appending to the file. Initially, the file pointer is at the beginning of the file. The method creates the file if it does not already exist.

You cannot have more than eight files open in the file system. This method returns the file identifier for the open file in `file_ID`. You use `file_ID` as the first argument to the other file I/O methods (such as `xpctarget.fs.fclose`, `xpctarget.fs.fread`, and `xpctarget.fs.fwrite`).

## Examples

Open the file `data.dat` in the target computer file system object `fsys`. Assign the resulting file handle to a variable for reading.

```
h = fopen(fsys,'data.dat') or fsys.fopen('data.dat')
ans =
    2883584
d = fread(h);
```

## See Also

`fopen` | `xpctarget.fs.fclose` | `xpctarget.fs.fread` | `xpctarget.fs.fwrite`

**Purpose** Read open target computer file

**Syntax** MATLAB command line

```
A = fread(file_obj,file_ID)
A = file_obj.fread(file_ID)
A = fread(file_obj, file_ID, offset, numbytes)
A = file_obj.fread(file_ID, offset, numbytes)
```

## Arguments

<code>file_obj</code>	Name of the xpctarget.fs object.
<code>file_ID</code>	File identifier of the file to read.
<code>offset</code>	Position from the beginning of the file from which fread can start to read.
<code>numbytes</code>	Maximum number of bytes fread can read.

## Description

Method of xpctarget.fs objects. From the host computer, `A = fread(file_obj,file_ID)` or `A = file_obj.fread(file_ID)` reads all the binary data from the file on the target computer and writes it into matrix A. The `file_ID` argument is the file identifier associated with an open file (see xpctarget.fs.fopen).

From the host computer, `A = fread(file_obj, file_ID, offset, numbytes)` or `A = file_obj.fread(file_ID, offset, numbytes)` reads a block of bytes from `file_ID` and writes the block into matrix A. The `offset` argument specifies the position from the beginning of the file from which this function can start to read. `numbytes` specifies the maximum number of bytes to read. To get a count of the total number of bytes read into A, use the following:

```
count = length(A);
```

# xpctarget.fs.fread

---

`length(A)` might be less than the number of bytes requested if that number of bytes are not currently available. It is zero if the operation reaches the end of the file.

## Examples

Open the file `data.dat` in the target computer file system object `fsys`. Assign the resulting file handle to a variable for reading.

```
h = fopen(fsys, 'data.dat') or fsys.fopen('data.dat')
ans =
    2883584
d = fread(h);
```

This reads the file `data.dat` and stores all of the contents of the file to `d`. This content is in the xPC Target file format.

## See Also

[fread](#) | [xpctarget.fs fclose](#) | [xpctarget.fs fopen](#) | [xpctarget.fs fwrite](#)



<b>Purpose</b>	Write binary data to open target computer file						
<b>Syntax</b>	<b>MATLAB command line</b>  <code>fwrite(file_obj,file_ID,A)</code> <code>file_obj.fwrite(file_ID,A)</code>						
<b>Arguments</b>	<table><tr><td><code>file_obj</code></td><td>Name of the <code>xpctarget.fs</code> object.</td></tr><tr><td><code>file_ID</code></td><td>File identifier of the file to write.</td></tr><tr><td><code>A</code></td><td>Elements of matrix <code>A</code> to be written to the specified file.</td></tr></table>	<code>file_obj</code>	Name of the <code>xpctarget.fs</code> object.	<code>file_ID</code>	File identifier of the file to write.	<code>A</code>	Elements of matrix <code>A</code> to be written to the specified file.
<code>file_obj</code>	Name of the <code>xpctarget.fs</code> object.						
<code>file_ID</code>	File identifier of the file to write.						
<code>A</code>	Elements of matrix <code>A</code> to be written to the specified file.						
<b>Description</b>	Method of <code>xpctarget.fs</code> objects. From the host computer, writes the elements of matrix <code>A</code> to the file identified by <code>file_ID</code> . The data is written to the file in column order. The <code>file_ID</code> argument is the file identifier associated with an open file (see <code>xpctarget.fs.fopen</code> ). <code>fwrite</code> requires that the file be open with write permission.						
<b>Examples</b>	<p>Open the file <code>data.dat</code> in the target computer file system object <code>fsys</code>. Assign the resulting file handle to a variable for writing.</p> <pre>h = fopen(fsys,'data.dat','w')</pre> <p>or</p> <pre>fsys.fopen('data.dat','w')</pre> <pre>ans =     2883584 d = fwrite(fsys,h,magic(5));</pre> <p>This writes the elements of matrix <code>A</code> to the file handle <code>h</code>. This content is written in column order.</p>						
<b>See Also</b>	<code>fwrite</code>   <code>xpctarget.fs.fclose</code>   <code>xpctarget.fs.fopen</code>   <code>xpctarget.fs.fread</code>						

# xpctarget.fs.getfilesize

---

**Purpose** Size of file on target computer

**Syntax** MATLAB command line

```
getfilesize(file_obj,file_ID)  
file_obj.getfilesize(file_ID)
```

**Arguments**

<code>file_obj</code>	Name of the <code>xpctarget.fs</code> object.
<code>file_ID</code>	File identifier of the file to get the size of.

**Description** Method of `xpctarget.fs` objects. From the host computer, gets the size (in bytes) of the file identified by the `file_ID` file identifier on the target computer file system. Use the xPC Target file object method `xpctarget.fs.fopen` to open the file system object.

**Examples** Get the size of the file identifier `h` for the file system object `fsys`.

```
getfilesize(fsys,h) or fsys.getfilesize(h)
```

**See Also** `xpctarget.fs.fopen`

**Purpose** Interpret raw data from xPC Target file format

**Syntax** `file=readxpcfile(data)`  
`readxpcfile('filename')`

**Arguments**

<code>data</code>	Vector of uint8 bytes.
<code>'filename'</code>	File from which the vector of uint8 bytes is read. Vector is written

**Description** The `readxpcfile` function converts xPC Target file format content (in bytes) to double precision data. A file scope creates the data.

The `readxpcfile` function returns a structure that contains the following fields:

- `version`  
Not used.
- `sector`  
Not used.
- `headersize`  
Not used.
- `numSignals`  
Array of signal names.
- `data`  
Array of signal data.
- `signalNames`  
Cell array of signal names.

After you download the data from a target computer, use one of the following to read the data:

# xpctarget.fs.readxpcfile

---

- To read the data after you download it to the target computer, use the `fread` function
- To download the data to the target computer and read it, use the `xpctarget.fs.fread` method.

`file=readxpcfile(data)` converts `data` to double precision data representing the signals and timestamps.

`readxpcfile('filename')` converts contents of `'filename'` to double precision data representing the signals and timestamps.

## Examples

Use the `xpctarget.fs` object to convert data:

```
file=xpctarget.fs;
h=file.fopen('filename');
data=file.fread(h);
file fclose(h);
file = readxpcfile(data);
```

Use the `xpctarget.ftp` object to copy the file from the target computer to the host computer, then read and convert the data.

```
xpcftp=xpctarget.ftp
xpcftp.get('filename')
datafile = readxpcfile('filename') % Convert the data
```

Use the `xpctarget.ftp` object to copy the file from the target computer to the host computer, then read and convert the data separately.

```
xpcftp=xpctarget.ftp
xpcftp.get('filename')
handle=fopen('filename')
data=fread(handle,'*uint8'); % Data should be read in uint8 format
fclose(handle);
data=data';
datafile = readxpcfile(data); % Convert the data
```

**See Also**

xpctarget.ftp.get (ftp) | xpctarget.fs.fopen |  
xpctarget.fs.fread

# xpctarget.fs.removefile

---

**Purpose** Remove file from target computer

**Syntax** MATLAB command line

```
removefile(file_obj,file_name)  
file_obj.removefile(file_name)
```

**Arguments**

file_name	Name of the file to remove from the target computer file system.
file_obj	Name of the xpctarget.fs object.

**Description** Method of xpctarget.fs objects. Removes a file from the target computer file system.

---

**Note** You cannot recover this file once it is removed.

---

**Examples** Remove the file data2.dat from the target computer file system fsys.

```
removefile(fsys,'data2.dat')
```

or

```
fsys.removefile('data2.dat')
```

**Purpose** Select target computer drive

**Syntax** **MATLAB command line**

```
selectdrive(file_obj, 'drive')  
file_obj.selectdrive('drive')
```

**Arguments**

drive	Name of the drive to set.
file_obj	Name of the xpctarget.fs object.

**Description** Method of xpctarget.fs objects. selectdrive sets the current drive of the target computer to the specified string. Enter the drive string with an extra backslash (\). For example, D:\\ for the D:\ drive.

---

**Note** Use the xpctarget.fsbases.cd method instead to get the same behavior.

---

**Examples** Set the current target computer drive to D:\.

```
selectdrive(fsys, 'D:\\')  
  
or  
  
fsys.selectdrive('D:\\')
```

# xpctarget.fsbase Class

---

**Purpose** Base class of file system and file transfer protocol (FTP) classes

**Description** This class is the base class for `xpctarget.fs` Class and `xpctarget.ftp` Class. All methods are inherited by the derived classes. The constructor for this class is called implicitly when the constructors for the derived classes are called:

## Methods

These methods are inherited by the derived classes.

Method	Description
<code>xpctarget.fsbase.cd</code>	Change folder on target computer
<code>xpctarget.fsbase.dir</code>	List contents of current folder on target computer
<code>xpctarget.fsbase.mkdir</code>	Make folder on target computer
<code>xpctarget.fsbase.pwd</code>	Current folder path of target computer
<code>xpctarget.fsbase.rmdir</code>	Remove folder from target computer



**Purpose** Change folder on target computer

**Syntax** MATLAB command line

```
cd(file_obj,target_PC_dir)
file_obj.cd(target_PC_dir)
```

**Arguments**

<code>file_obj</code>	Name of the <code>xpctarget.ftp</code> or <code>xpctarget.fs</code> object.
<code>target_PC_dir</code>	Name of the target computer folder to change to.

**Description** Method of `xpctarget.fsbase`, `xpctarget.ftp`, and `xpctarget.fs` objects. From the host computer, changes folder on the target computer.

**Examples** Change folder from the current to one named `logs` for the file system object `fsys`.

```
cd(fsys,logs) or fsys.cd(logs)
```

Change folder from the current to one named `logs` for the FTP object `f`.

```
cd(f,logs) or f.cd(logs)
```

**See Also** `cd` | `xpctarget.fsbase.mkdir` | `xpctarget.fsbase.pwd`

# xpctarget.fsbase.dir

---

**Purpose** List contents of current folder on target computer

**Syntax** MATLAB command line

```
dir(file_obj)
```

**Arguments** `file_obj` Name of the `xpctarget.ftp` or `xpctarget.fs` object.

**Description** Method of `xpctarget.fsbase`, `xpctarget.ftp`, and `xpctarget.fs` objects. From the host computer, lists the contents of the current folder on the target computer.

To get the results in an M-by-1 structure, use a syntax like `ans=dir(file_obj)`. This syntax returns a structure like the following:

```
ans =  
1x5 struct array with fields:  
name  
date  
time  
bytes  
isdir
```

where

- `name` — Name of an object in the folder, shown as a cell array. The name, stored in the first element of the cell array, can have up to eight characters. The three-character file extension is stored in the second element of the cell array.
- `date` — Date of the last save of that object
- `time` — Time of the last save of that object
- `bytes` — Size in bytes of that object
- `isdir` — Logical value indicating that the object is (1) or is not (0) a folder

## Examples

List the contents of the current folder for the file system object fsys.  
You can also list the contents of the current folder for the FTP object f.

```
dir(fsys) or dir(f)
4/12/1998    20:00                222390      IO  SYS
 11/2/2003   13:54                 6      MSDOS  SYS
 11/5/1998   20:01                93880  COMMAND  COM
 11/2/2003   13:54 <DIR>                0      TEMP
 11/2/2003   14:00                 33  AUTOEXEC  BAT
  11/2/2003  14:00                512  BOOTSECT  DOS
  18/2/2003  16:33                4512  SC1SIGNA  DAT
 18/2/2003   16:17 <DIR>                0      FOUND  000
 29/3/2003   19:19                8512      DATA  DAT
 28/3/2003   16:41                8512  DATADATA  DAT
 28/3/2003   16:29                4512  SC4INTEG  DAT
  1/4/2003    9:28             201326592  PAGEFILE  SYS
 11/2/2003   14:13 <DIR>                0      WINNT
  4/5/2001   13:05             214432  NTLDR      '
  4/5/2001   13:05             34468  NTDETECT  COM
 11/2/2003   14:15 <DIR>                0  DRIVERS
  22/1/2001  11:42                217    BOOT    INI '
 28/3/2003   16:41                8512      A      DAT
 29/3/2003   19:19                2512  SC3SIGNA  DAT
 11/2/2003   14:25 <DIR>                0  INETPUB
 11/2/2003   14:28                0    CONFIG  SYS
 29/3/2003   19:10                2512  SC3INTEG  DAT
  1/4/2003   18:05                2512  SC1GAIN  DAT
  11/2/2003  17:26 <DIR>                0  UTILIT~1
```

You must use the `dir(f)` syntax to list the contents of the folder.

## See Also

```
dir | xpctarget.fsbase.mkdir | xpctarget.fsbase.cd |
xpctarget.fsbase.pwd
```

# xpctarget.fsbase.mkdir

---

**Purpose**            Make folder on target computer

**Syntax**            **MATLAB command line**

```
mkdir(file_obj,dir_name)
file_obj.mkdir(dir_name)
```

**Arguments**

<code>file_obj</code>	Name of the <code>xpctarget.ftp</code> or <code>xpctarget.fs</code> object.
<code>dir_name</code>	Name of the folder to be created.

**Description**      Method of `xpctarget.fsbase`, `xpctarget.ftp`, and `xpctarget.fs` objects. From the host computer, makes a new folder in the current folder on the target computer file system.

Note that to delete a folder from the target computer, you need to reboot the PC into DOS or some other operating system and use a utility in that system to delete the folder.

**Examples**            Create a new folder, `logs`, in the target computer file system object `fsys`.

```
mkdir(fsys,logs)
```

or

```
fsys.mkdir(logs)
```

Create a new folder, `logs`, in the target computer FTP object `f`.

```
mkdir(f,logs) or f.mkdir(logs)
```

**See Also**            `mkdir` | `xpctarget.fsbase.dir` | `xpctarget.fsbase.pwd`

<b>Purpose</b>	Current folder path of target computer
<b>Syntax</b>	<b>MATLAB command line</b>  pwd(file_obj) file_obj.pwd
<b>Arguments</b>	file_obj      Name of the xpctarget.ftp or xpctarget.fs object.
<b>Description</b>	Method of xpctarget.fsbased, xpctarget.ftp, and xpctarget.fs objects. Returns the pathname of the current target computer folder.
<b>Examples</b>	Return the target computer current folder for the file system object fsys.  pwd(fsys) or fsys.pwd  Return the target computer current folder for the FTP object f.  pwd(f) or f.pwd
<b>See Also</b>	pwd   xpctarget.fsbased.dir   xpctarget.fsbased.mkdir

# xpctarget.fsbase.rmdir

---

**Purpose** Remove folder from target computer

**Syntax** MATLAB command line

```
rmdir(file_obj,dir_name)
file_obj.rmdir(dir_name)
```

**Arguments**

dir_name	Name of the folder to remove from the target computer file system.
file_obj	Name of the xpctarget.fs object.

**Description** Method of xpctarget.fsbase, xpctarget.ftp, and xpctarget.fs objects. Removes a folder from the target computer file system.

---

**Note** You cannot recover this folder once it is removed.

---

**Examples** Remove the folder data2dir.dat from the target computer file system fsys.

```
rmdir(f,'data2dir.dat')
```

or

```
fsys.rmdir('data2dir.dat')
```

**Purpose** Manage the directories and files on the target computer via file transfer protocol (FTP)

**Description** The FTP object represents the file on the target computer. You work with the file directories using the inherited methods, and transport the file between the host and target computers using the `xpctarget.ftp` methods.

## Constructor

Constructor	Description
<code>xpctarget.ftp</code>	Create file transfer protocol (FTP) object

## Methods

These methods are inherited from `xpctarget.fsbase` Class.

Method	Description
<code>xpctarget.fsbase.cd</code>	Change folder on target computer
<code>xpctarget.fsbase.dir</code>	List contents of current folder on target computer
<code>xpctarget.fsbase.mkdir</code>	Make folder on target computer
<code>xpctarget.fsbase.pwd</code>	Current folder path of target computer
<code>xpctarget.fsbase.rmdir</code>	Remove folder from target computer

These methods are specific to class `ftp`.

Method	Description
<code>xpctarget.ftp.get(ftp)</code>	Retrieve copy of requested file from target computer
<code>xpctarget.ftp.put</code>	Copy file from host computer to target computer

# xpctarget.ftp

---

**Purpose** Create file transfer protocol (FTP) object

**Syntax** MATLAB command line

```
file_object = xpctarget.ftp('mode', 'arg1', 'arg2')
```

**Arguments**

`file_object` Variable name to reference the FTP object.

`mode` Optionally, enter the communication mode:

---

**Note** RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.

---

TCPIP Specify TCP/IP connection with target computer.

RS232 Specify RS-232 connection with target computer.

`arg1` Optionally, enter an argument based on the `mode` value:

IP address If `mode` is 'TCPIP', enter the IP address of the target computer.

COM port If `mode` is 'RS232', enter the host COM port.

`arg2` Optionally, enter an argument based on the `mode` value:

Port If `mode` is 'TCPIP', enter the port number for the target computer.

Baud rate If `mode` is 'RS232', enter the baud rate for the connection between the host and target computer.



**Description**

Constructor of an FTP object (`xpctarget.ftp` Class). The FTP object represents the file on the target computer. You work with the file by changing the file object using methods.

If you have one target computer object, or if you designate a target computer as the default one in your system, use the syntax

```
file_object=xpctarget.ftp
```

If you have multiple target computers in your system, or if you want to identify a target computer with the file object, use the following syntax to create the additional file objects.

```
file_object=xpctarget.ftp('mode', 'arg1', 'arg2')
```

**Examples**

In the following example, a file object for a target computer with an RS-232 connection is created.

```
f=xpctarget.ftp('RS232', 'COM1', '115200')
```

```
f =  
xpctarget.ftp
```

Optionally, if you have an `xpctarget.xpc` object, you can construct an `xpctarget.ftp` object by passing the `xpctarget.xpc` object variable to the `xpctarget.ftp` constructor as an argument.

```
>> tg1=xpctarget.xpc('RS232', 'COM1', '115200');  
>> f2=xpctarget.ftp(tg1)
```

```
f2 =  
  
xpctarget.ftp
```

# xpctarget.ftp.get (ftp)

---

**Purpose** Retrieve copy of requested file from target computer

**Syntax** MATLAB command line

```
get(file_obj,file_name)
file_obj.get(file_name)
```

**Arguments**

<code>file_obj</code>	Name of the <code>xpctarget.ftp</code> object.
<code>file_name</code>	Name of a file on the target computer.

**Description** Method of `xpctarget.ftp` objects. Copies the specified filename from the target computer to the current folder of the host computer. `file_name` must be either a fully qualified filename on the target computer, or located in the current folder of the target computer.

**Examples** Retrieve a copy of the file named `data.dat` from the current folder of the target computer file object `f`.

```
get(f,'data.dat') or f.get('data.dat')
ans = data.dat
```

**See Also** `xpctarget.ftp.put`

**Purpose** Copy file from host computer to target computer

**Syntax** MATLAB command line

```
put(file_obj,file_name)
file_obj.put(file_name)
```

**Arguments**

<code>file_obj</code>	Name of the <code>xpctarget.ftp</code> object.
<code>file_name</code>	Name of the file to copy to the target computer.

**Description** Method of `xpctarget.ftp` objects. Copies a file from the host computer to the target computer. `file_name` must be a file in the current folder of the host computer. The method writes `file_name` to the target computer disk.

`put` might be slower than the `get` operation for the same file. This is expected behavior.

**Examples** Copy the file `data2.dat` from the current folder of the host computer to the current folder of the target computer FTP object `f`.

```
put(f,'data2.dat')
```

or

```
fsys.put('data2.dat')
```

**See Also** `xpctarget.fsbase.dir` | `xpctarget.ftp.get` (`ftp`)

# xpctarget.targets Class

---

**Purpose** Container object to manage target computer environment collection objects

**Description** The targets class contains a collection of environment settings, stored in xpctarget.env Class objects.

## Constructor

Constructor	Description
xpctarget.targets	Create container object to manage target computer environment collection objects

## Methods

Method	Description
xpctarget.targets.Add (env collection object)	Add a new xPC Target environment collection object.
xpctarget.targets.getTargetNames (env collection object)	Retrieve all xPC Target environment collection object names.
xpctarget.targets.Item (env collection object)	Retrieve xPC Target environment collection object.
xpctarget.targets.makeDefault (env collection object)	Set target computer environment collection object as default.
xpctarget.targets.Remove (env collection object)	Remove an xPC Target environment collection object.

## Properties

Property	Description	Writable
CCompiler	<p><b>Note</b></p> <ul style="list-style-type: none"><li>• CCompiler and CompilerPath will be removed in a future version. Use the xpcsetCC function to configure the compiler instead.</li><li>• WATCOM compiler support will be removed in a future release. Use a Microsoft compiler instead.</li></ul> <hr/> <p>Values are 'Watcom' and 'VisualC'.</p>	Yes
CompilerPath	Value is a valid compiler root folder. Enter the path where you installed a Watcom C/C++ or Microsoft Visual Studio C/C++ compiler.	Yes
DefaultTarget	Returns an xpctarget.env object that references the default target computer object environment.	No
NumTargets	Returns the number of target computer environment objects in the container.	No

# xpctarget.targets

---

**Purpose** Create container object to manage target computer environment collection objects

**Syntax** **MATLAB command line**  
`env_collection_object = xpctarget.targets`

**Description** Constructor for target environment object collection (`xpctarget.targets` Class). The collection manages the environment object (`xpctarget.env` Class) for a multitarget xPC Target system. (This is in contrast to the `setxpcenv` and `getxpcenv` functions, which manage the environment properties for the default target computer.) You work with the environment objects by changing the environment properties using methods.

Use the syntax

```
env_object = xpctarget.targets
```

Access properties of an `env_collection_object` object with `env_collection_object.propertyname`, `env_collection_object.propertyname.propertyname`, or with the `xpctarget.targets.get (env collection object)` and `xpctarget.targets.set (env collection object)` commands.

Access an individual environment object via `xpctarget.targets.Item (env collection object)`,

**Examples** Create an environment container object. With this object, you can manage the environment collection objects for all the targets in your system.

```
tgs=xpctarget.targets
```

**See Also** `xpctarget.targets.get (env collection object)` | `xpctarget.targets.set (env collection object)`

# xpctarget.targets.Add (env collection object)

---

<b>Purpose</b>	Add new xPC Target environment collection object
<b>Syntax</b>	<b>MATLAB command line</b>  env_collection_object.Add
<b>Description</b>	Method of xpctarget.targets objects. Add creates an xPC Target environment collection object on the host computer.
<b>Examples</b>	<p>Add a new xPC Target environment collection object to the system. Assume that tgs represents the environment collection object. The first get(tgs) function returns the current number of target computers. The second function returns the number of target computers after you add one.</p> <pre>tgs=xpctarget.targets; get(tgs); tgs.Add; get(tgs);</pre>
<b>See Also</b>	xpctarget.targets   xpctarget.targets.set (env collection object)   xpctarget.targets.get (env collection object)

# xpctarget.targets.get (env collection object)

**Purpose** Return target object collection environment property values

**Syntax** MATLAB command line

```
get(env_collection_object, 'env_collection_object_property')
```

**Arguments**

env_collection_object	Name of a collection of target objects.
'env_collection_object_property'	Name of a target object environment property.

**Description** get gets the values of environment properties for a collection of target objects.

The environment properties for a target environment object collection 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.

Property	Description	Writable
CCompiler	<p><b>Note</b></p> <ul style="list-style-type: none"><li>• CCompiler and CompilerPath will be removed in a future version. Use the xpcgetCC function to configure the compiler instead.</li><li>• WATCOM compiler support will be removed in a future release. Use a Microsoft compiler instead.</li></ul> <p>Values are 'Watcom' and 'VisualC'. From the xPC Target Explorer</p>	Yes



# xpctarget.targets.get (env collection object)

Property	Description	Writable
	window compiler list, select either Watcom or VisualC.	
CompilerPath	Value is a valid compiler root folder. Enter the path where you installed a Watcom C/C++ or Microsoft Visual Studio C/C++ compiler.  If the path is invalid or the folder does not contain the compiler, an error message appears when you build a target application.	Yes
DefaultTarget	Contains an instance of the default target environment object (xpctarget.env).	No
NumTargets	Contains the number of target objects in the xPC Target system. Note that this is not the actual number of target computers in the system.	No

## Examples

List the values of all the target object collection environment property values. Assume that `tgs` represents the target object collection environment.

```
tgs=xpctarget.targets;  
get(tgs);
```

List the value for the target object environment collection property `NumTargets`. Note that the property name is a string, in quotation marks, and not case sensitive.

```
get(tgs, 'NumTargets') or tgs.get('NumTargets')
```

## See Also

```
get | xpctarget.targets.set (env collection object) | set
```

# xpctarget.targets.getTargetNames (env collection object)

---

**Purpose** Retrieve xPC Target environment object names

**Syntax** MATLAB command line  
`env_collection_object.getTargetNames`

**Description** Method of `xpctarget.targets` objects. `getTargetNames` retrieves the names of all existing xPC Target environment collection objects from the `xpctarget.targets` class.

**Examples** Retrieve the names of all xPC Target environment collection objects in the system. Assume that `tgs` represents the target object collection environment.

```
tgs=xpctarget.targets;  
get(tgs);  
tgs.getTargetNames
```

**See Also** `xpctarget.targets` | `xpctarget.targets.set` (env collection object) | `xpctarget.targets.get` (env collection object)

# xpctarget.targets.Item (env collection object)

---

<b>Purpose</b>	Retrieve specific xPC Target environment (env) object
<b>Syntax</b>	<b>MATLAB command line</b>  <code>env_collection_object.Item('env_object_name')</code>
<b>Description</b>	Method of <code>xpctarget.targets</code> objects. <code>Item</code> retrieves a specific environment object ( <code>xpctarget.env Class</code> ) from the <code>xpctarget.targets</code> class. Use this method to work with a particular target computer environment object.
<b>Examples</b>	Retrieve a new xPC Target environment collection object from the system. Assume that <code>tgs</code> represents the target object collection environment.  <pre>tgs=xpctarget.targets; get(tgs); tgs.getTargetNames tgs.Item('TargetPC1')</pre>
<b>See Also</b>	<code>xpctarget.targets</code>   <code>xpctarget.targets.set (env collection object)</code>   <code>xpctarget.targets.get (env collection object)</code>

# xpctarget.targets.makeDefault (env collection object)

---

**Purpose** Set specific target computer environment object as default

**Syntax** MATLAB command line  
`env_collection_object.makeDefault('env_object_name')`

**Description** Method of `xpctarget.targets` objects. `makeDefault` sets the specified target computer environment object as the default target computer from the `xpctarget.targets` class.

**Examples** Set the specified target collection object as the default target computer collection. Assume that `tgs` represents the target object collection environment.

```
tgs=xpctarget.targets;  
get(tgs);  
tgs.getTargetNames  
tgs.makeDefault('TargetPC2')
```

**See Also** `xpctarget.targets` | `xpctarget.targets.set` (env collection object) | `xpctarget.targets.get` (env collection object)

# xpctarget.targets.Remove (env collection object)

---

**Purpose** Remove specific xPC Target environment object

**Syntax** MATLAB command line

```
env_collection_object.Remove('env_collection_object_name')
```

**Description** Method of xpctarget.targets objects. Remove removes an existing xPC Target environment object from the environment collection. Note that if you remove the target environment object of the default target computer, the next target environment object becomes the default target computer. You can remove all but the last target computer, which becomes the default target computer.

**Examples** Remove an xPC Target environment collection object from the system. Assume that tgs represents the target object collection environment.

```
tgs=xpctarget.targets;  
get(tgs);  
tgs.getTargetNames  
tgs.Remove('TargetPC2')
```

**See Also** xpctarget.targets | xpctarget.targets.set (env collection object) | xpctarget.targets.get (env collection object)

# xpctarget.targets.set (env collection object)

---

**Purpose** Change target object environment collection object property values

**Syntax** MATLAB command line

```
set(env_collection_object)
set(env_collection_object, 'property_name1',
'property_value1', 'property_name2', 'property_value2', . . .)
env_collection_object.set('property_name1',
'property_value1')
set(env_collection_object, property_name_vector,
property_value_vector)
env_collection_object.property_name = property_value
```

<b>Arguments</b>	<code>env_collection_object</code>	Name of a target environment collection object.
	<code>'property_name'</code>	Name of a target object environment collection property. Always use quotation marks.
	<code>property_value</code>	Value for a target object environment collection property. Always use quotation marks for character strings; quotation marks are optional for numbers.

**Description** `set` sets the values of environment properties for a collection of target object environments. Not all properties are user writable.

Properties must be entered in pairs or, using the alternative syntax, as one-dimensional cell arrays of the same size. This means they must 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 environment properties for a target object collection 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.

## xpctarget.targets.set (env collection object)

Property	Description	Writable
CCompiler	<p><b>Note</b></p> <ul style="list-style-type: none"><li>• CCompiler and CompilerPath will be removed in a future version. Use the xpcsetCC function to configure the compiler instead.</li><li>• WATCOM compiler support will be removed in a future release. Use a Microsoft compiler instead.</li></ul> <p>Values are 'Watcom' and 'VisualC'. From the xPC Target Explorer window compiler list, select either Watcom or VisualC.</p>	Yes
CompilerPath	<p>Value is a valid compiler root folder. Enter the path where you installed a Watcom C/C++ or Microsoft Visual Studio C/C++ compiler.</p> <p>If the path is invalid or the folder does not contain the compiler, an error message appears when you build a target application.</p>	Yes
DefaultTarget	<p>Contains an instance of the default target environment object (xpctarget.env).</p>	No
NumTargets	<p>Contains the number of target objects in the xPC Target system. Note that this is not the actual number of target computers in the system.</p>	No

## **xpctarget.targets.set (env collection object)**

---

### **See Also**

`get` | `set` | `xpctarget.targets.get` (env collection object)



**Purpose** Target object representing target application

**Description** Provides access to methods and properties used to start and stop the target application, read and set parameters, monitor signals, and retrieve status information about the target computer.

## Constructor

Constructor	Description
xpctarget.xpc	Create target object representing target application

## Methods

Method	Description
xpctarget.xpc.addscope	Create scopes
xpctarget.xpc.close	Close serial port connecting host computer with target computer
xpctarget.xpc.get (target application object)	Return target application object property values
xpctarget.xpc.getlog	All or part of output logs from target object
xpctarget.xpc.getparam	Value of target object parameter index
xpctarget.xpc.getparam	Parameter index from parameter list
xpctarget.xpc.getparam	Block path and parameter name from index list
xpctarget.xpc.getscope	Scope object pointing to scope defined in kernel
xpctarget.xpc.getsignal	Value of target object signal index
xpctarget.xpc.getsignal	Signal index or signal property from signal list
xpctarget.xpc.getsignal	Return label of signal indices
xpctarget.xpc.getsignal	Return signal label
xpctarget.xpc.getsignal	Signal name from index list

# xpctarget.xpc Class

Method	Description
<code>xpctarget.xpc.getxpcpci</code>	Determine which PCI boards are installed in target computer
<code>xpctarget.xpc.load</code>	Download target application to target computer
<code>xpctarget.xpc.loadparameters</code>	Restore parameter values saved in specified file
<code>xpctarget.xpc.reboot</code>	Reboot target computer
<code>xpctarget.xpc.remscope</code>	Remove scope from target computer
<code>xpctarget.xpc.saveparameters</code>	Save current target application parameter values
<code>xpctarget.xpc.set</code> (target application object)	Change target application object property values
<code>xpctarget.xpc.setparameters</code>	Change writable target object parameters
<code>xpctarget.xpc.start</code> (target application object)	Start execution of target application on target computer
<code>xpctarget.xpc.stop</code> (target application object)	Stop execution of target application on target computer
<code>xpctarget.xpc.targetping</code>	Test communication between host and target computers
<code>xpctarget.xpc.unload</code>	Remove current target application from target computer

## Properties

Properties are read using `xpctarget.xpc.get` (target application object). Writable properties are written using `xpctarget.xpc.set` (target application object).

Property	Description	Writable
Application	Name of the Simulink model and target application built from that model.	No
AvgTET	<p>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.</p> <p>The TET includes:</p> <ul style="list-style-type: none"> <li>• Complete I/O latency.</li> <li>• Data logging (the parts that happen in a real-time task). This includes data captured in scopes.</li> <li>• Asynchronous interruptions.</li> <li>• Parameter updating latency (if the <b>Double buffer parameter changes</b> parameter in the <b>xPC Target options</b> node using the model Simulation &gt; Configuration Parameters dialog box).</li> </ul> <p>Note that the TET is not the only consideration in determining the minimum achievable sample time. Other considerations, not included in the TET, are:</p> <ul style="list-style-type: none"> <li>• Time required to measure TET</li> <li>• Interrupt latency required to schedule and run one step of the model</li> </ul>	No

## xpctarget.xpc Class

---

Property	Description	Writable
CommunicationTimeOut	Communication timeout between host and target computer, in seconds.	Yes
Connected	Communication status between the host computer and the target computer. Values are 'Yes' and 'No'.	No
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. Returning this status to 'none' requires either a faster processor or a larger sample time.	No
ExecTime	Execution time. Time, in seconds, since your target application started running. When the target application stops, the total execution time is displayed.	No
LogMode	Controls which data points are logged: <ul style="list-style-type: none"><li>• Time-equidistant logging. Logs a data point at every time interval. Set value to 'Normal'.</li><li>• Value-equidistant logging. Logs a data point only when an output signal from the <code>OutputLog</code> changes by a specified value (increment). Set the value to the difference in signal values.</li></ul>	Yes

Property	Description	Writable
MaxLogSamples	<p>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.</p> <p>This value is calculated by dividing the <b>Signal Logging Buffer Size</b> by the number of logged signals. The <b>Signal Logging Buffer Size</b> box is located at <b>Simulation</b> menu <b>Configuration Parameters &gt; xPC Target options</b> pane.</p>	No
MaxTET	<p>Maximum task execution time. Corresponds to the slowest time (longest time measured), in seconds, to update model equations and post outputs.</p>	No
MinTET	<p>Minimum task execution time. Corresponds to the fastest time (smallest time measured), in seconds, to update model equations and post outputs.</p>	No
Mode	<p>Type of Simulink Coder code generation. Values are 'Real-Time Singletasking', 'Real-Time Multitasking', and 'Accelerate'. The default value is 'Real-Time Singletasking'. 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.</p>	No

# xpctarget.xpc Class

Property	Description	Writable
NumLogWraps	The number of times the circular buffer wrapped. The buffer wraps each time the number of samples exceeds MaxLogSamples.	No
NumParameters	The number of parameters from your Simulink model that you can tune or change.	No
NumSignals	The number of signals from your Simulink model that are available to be viewed with a scope.	No
OutputLog	Storage in the MATLAB workspace for the output or Y-vector logged during execution of the target application.	No
Parameters	List of tunable parameters. This list is visible only when ShowParameters is set to 'on': <ul style="list-style-type: none"><li>• Property value. Value of the parameter in a Simulink block.</li><li>• Type. Data type of the parameter. Always double.</li><li>• Size. Size of the parameter. For example, scalar, 1-by-2 vector, or 2-by-3 matrix.</li><li>• Parameter name. Name of a parameter in a Simulink block.</li><li>• Block name. Name of a Simulink block.</li></ul>	No

Property	Description	Writable
SampleTime	Time between samples. This value equals the step size, in seconds, for updating the model equations and posting the outputs. (See “User Interaction” in the <i>xPC Target Getting Started Guide</i> for limitations on target property changes to sample times.)	Yes
Scopes	List of index numbers, with one index for each scope.	No
SessionTime	Time since the kernel started running on your target computer. This is also the elapsed time since you booted the target computer. Values are in seconds.	No
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' and 'off'.	Yes
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' and 'off'.	Yes
Signals	List of viewable signals. This list is visible only when ShowSignals is set to 'on'. <ul style="list-style-type: none"> <li>• Property name. S0, S1. . .</li> <li>• Property value. Value of the signal.</li> <li>• Block name. Name of the Simulink block the signal is from.</li> </ul>	No

## xpctarget.xpc Class

Property	Description	Writable
StateLog	Storage in the MATLAB workspace for the state or x-vector logged during execution of the target application.	No
Status	Execution status of your target application. Values are 'stopped' and 'running'.	No
StopTime	Time when the target application stops running. Values are in seconds. The original value is set in the <b>Simulation</b> menu <b>Configuration Parameters</b> dialog.  When the ExecTime reaches StopTime, the application stops running.	Yes
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 select the <b>Log Task Execution Time</b> check box located at <b>Simulation</b> menu <b>Configuration Parameters &gt; xPC Target options</b> pane.	No
TimeLog	Storage in the MATLAB workspace for the time or T-vector logged during execution of the target application.	No
ViewMode	Display either all scopes or a single scope on the target computer. Value is 'all' or a single scope index. This property is active only if the environment property TargetScope is set to enabled.	Yes



**Purpose** Create target object representing target application

**Syntax** MATLAB command line

```
target_object = xpctarget.xpc('mode', 'arg1', 'arg2')
target_object=xpctarget.xpc('target_object_name')
```

**Arguments**

target_object	Variable name to reference the target object
mode	Optionally, enter the communication mode

---

**Note** RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.

---

	TCP/IP	Enable TCP/IP connection with target computer.
	RS232	Enable RS-232 connection with target computer.
arg1	Optionally, enter an argument based on the mode value:	
	IP address	If mode is 'TCP/IP', enter the IP address of the target computer.
	COM port	If mode is 'RS232', enter the host COM port.
arg2	Optionally, enter an argument based on the mode value:	
	Port	If mode is 'TCP/IP', enter the port number for the target computer.

Baud rate      If mode is 'RS232', enter the baud rate for the connection between the host and target computer.

target\_object\_name Target object name as specified in the xPC Target Explorer

## Description

Constructor of a target object (`xpctarget.xpc` Class). The target object represents the target application and target computer. You make changes to the target application by changing the target object using methods and properties.

If you have one target computer, or if you designate a target computer as the default one in your system, use the syntax

```
target_object=xpctarget.xpc
```

If you have multiple target computers in your system, use the following syntax to create the additional target objects.

```
target_object=xpctarget.xpc('mode', 'arg1', 'arg2')
```

If you have a target computer object in the xPC Target Explorer, you can use the following syntax to construct a corresponding target object from the MATLAB Command Window.

```
target_object=xpctarget.xpc('target_object_name')
```

## Examples

Before you build a target application, you can check the connection between your host and target computers by creating a target object, then using the `xpctarget.xpc.targetping` method to check the connection.

```
tg = xpctarget.xpc
xPC Object
    Connected                    = Yes
    Application                 = loader
```

```
tg.targetping
```

```
ans =  
  
success
```

If you have a second target computer for which you want to check the connection, create a second target object. In the following example, the connection with the second target computer is an RS-232 connection.

```
tg1=xpctarget.xpc('RS232','COM1','115200')  
  
xPC Object  
  Connected          = Yes  
  Application        = loader
```

If you have an xPC Target Explorer target object, and you want to construct a corresponding target object in the MATLAB Command Window, use a command like the following:

```
target_object=xpctarget.xpc('TargetPC1')
```

**See Also**

```
xpctarget.xpc.get (target application object) |  
xpctarget.xpc.set (target application object) |  
xpctarget.xpc.targetping
```

# xpctarget.xpc.addscope

---

**Purpose** Create scopes

**Syntax** **MATLAB command line**

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

```
addscope(target_object, scope_type, scope_number)
target_object.addscope(scope_type, scope_number)
```

Create a scope, scope object, and assign to a MATLAB variable

```
scope_object = addscope(target_object,
    scope_type, scope_number)
scope_object = target_object.addscope(scope_type,
    scope_number)
```

**Target computer command line** — When you are using this command on the target computer, you can only add a target scope.

```
addscope
addscope scope_number
```

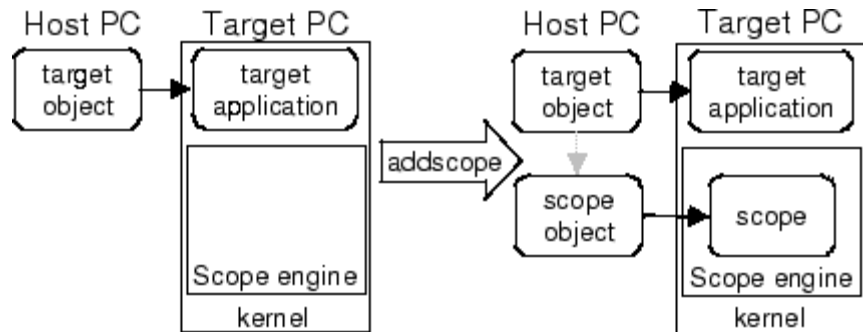
**Arguments**

- target\_object** Name of a target object. The default target name is `tg`.
- scope\_type** Values are `'host'`, `'target'`, or `'file'`. This argument is optional with `host` as the default value.
- scope\_number** Vector of new scope indices. This argument is optional. 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**

`addscope` creates a scope of the specified type 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. The xPC Target product supports 10 target or host scopes, and eight file scopes, for a maximum of 28 scopes. If you try to add a scope with the same index as an existing scope, the result is an error.



## Examples

Create a scope and scope object `sc1` using the method `addscope`. A target scope is created on the target computer with an index of 1, and a scope object is created on the host computer, 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 create a scope object, corresponding to this scope, using the method `getscope`. A target scope is created on the target computer with an index of 1, and a scope object is created on the host computer, 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)
```

# xpctarget.xpc.addscope

---

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

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

Create a scope and scope object `sc4` of type `file` using the method `addscope`. A file scope is created on the target computer with an index of 4. A scope object is created on the host computer and is assigned to the variable `sc4`. The target object property `Scopes` is changed from `No scopes defined` to `4`.

```
sc4 = addscope(tg, 'file', 4) or sc4 = tg.addscope('file', 4)
```

## See Also

`xpctarget.xpc.remscope` | `xpctarget.xpc.getscope`

## How To

- “Selected Examples”

**Purpose** Close serial port connecting host computer with target computer

**Syntax** MATLAB command line

```
close(target_object)  
target_object.close
```

**Arguments** target\_object Name of a target object.

**Description** close closes the serial connection between the host computer and a target computer. If you want to use the serial port for another function without quitting the MATLAB window – for example, a modem – use this function to close the connection.

# xpctarget.xpc.get (target application object)

---

**Purpose** Return target application object property values

**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** `get` gets the value of readable target object properties from a target object.

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.

Property	Description	Writable
Application	Name of the Simulink model and target application built from that model.	No
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.  The TET includes: <ul style="list-style-type: none"><li>• Complete I/O latency.</li><li>• Data logging (the parts that happen in a real-time task). This includes data captured in scopes.</li></ul>	No



## xpctarget.xpc.get (target application object)

Property	Description	Writable
	<ul style="list-style-type: none"><li>Asynchronous interruptions.</li><li>Parameter updating latency (if the <b>Double buffer parameter changes</b> parameter in the <b>xPC Target options</b> node using the model Simulation &gt; Configuration Parameters dialog box).</li></ul> <p>Note that the TET is not the only consideration in determining the minimum achievable sample time. Other considerations, not included in the TET, are:</p> <ul style="list-style-type: none"><li>Time required to measure TET</li><li>Interrupt latency required to schedule and run one step of the model</li></ul>	
CommunicationTimeout	Communication timeout between host and target computer, in seconds.	Yes
Connected	Communication status between the host computer and the target computer. Values are 'Yes' and 'No'.	No
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. Returning this status to 'none' requires either a faster processor or a larger sample time.	No
ExecTime	Execution time. Time, in seconds, since your target application started running. When the target application stops, the total execution time is displayed.	No

## xpctarget.xpc.get (target application object)

Property	Description	Writable
LogMode	Controls which data points are logged: <ul style="list-style-type: none"><li>• Time-equidistant logging. Logs a data point at every time interval. Set value to 'Normal'.</li><li>• Value-equidistant logging. Logs a data point only when an output signal from the OutputLog changes by a specified value (increment). Set the value to the difference in signal values.</li></ul>	Yes
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 <b>Signal Logging Buffer Size</b> by the number of logged signals. The <b>Signal Logging Buffer Size</b> box is located at <b>Simulation</b> menu <b>Configuration Parameters &gt; xPC Target options</b> pane.	No
MaxTET	Maximum task execution time. Corresponds to the slowest time (longest time measured), in seconds, to update model equations and post outputs.	No
MinTET	Minimum task execution time. Corresponds to the fastest time (smallest time measured), in seconds, to update model equations and post outputs.	No

## xpctarget.xpc.get (target application object)

Property	Description	Writable
Mode	Type of Simulink Coder code generation. Values are 'Real-Time Singletasking', 'Real-Time Multitasking', and 'Accelerate'. The default value is 'Real-Time Singletasking'. 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.	No
NumLogWraps	The number of times the circular buffer wrapped. The buffer wraps each time the number of samples exceeds MaxLogSamples.	No
NumParameters	The number of parameters from your Simulink model that you can tune or change.	No
NumSignals	The number of signals from your Simulink model that are available to be viewed with a scope.	No
OutputLog	Storage in the MATLAB workspace for the output or Y-vector logged during execution of the target application.	No

## xpctarget.xpc.get (target application object)

Property	Description	Writable
Parameters	<p>List of tunable parameters. This list is visible only when ShowParameters is set to 'on':</p> <ul style="list-style-type: none"><li>• Property value. Value of the parameter in a Simulink block.</li><li>• Type. Data type of the parameter. Always double.</li><li>• Size. Size of the parameter. For example, scalar, 1-by-2 vector, or 2-by-3 matrix.</li><li>• Parameter name. Name of a parameter in a Simulink block.</li><li>• Block name. Name of a Simulink block.</li></ul>	No
SampleTime	<p>Time between samples. This value equals the step size, in seconds, for updating the model equations and posting the outputs. (See “User Interaction” in the <i>xPC Target Getting Started Guide</i> for limitations on target property changes to sample times.)</p>	Yes
Scopes	<p>List of index numbers, with one index for each scope.</p>	No
SessionTime	<p>Time since the kernel started running on your target computer. This is also the elapsed time since you booted the target computer. Values are in seconds.</p>	No
ShowParameters	<p>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' and 'off'.</p>	Yes

## xpctarget.xpc.get (target application object)

Property	Description	Writable
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' and 'off'.	Yes
Signals	List of viewable signals. This list is visible only when ShowSignals is set to 'on'. <ul style="list-style-type: none"><li>• Property name. S0, S1. . .</li><li>• Property value. Value of the signal.</li><li>• Block name. Name of the Simulink block the signal is from.</li></ul>	No
StateLog	Storage in the MATLAB workspace for the state or x-vector logged during execution of the target application.	No
Status	Execution status of your target application. Values are 'stopped' and 'running'.	No
StopTime	Time when the target application stops running. Values are in seconds. The original value is set in the <b>Simulation</b> menu <b>Configuration Parameters</b> dialog.  When the ExecTime reaches StopTime, the application stops running.	Yes

## xpctarget.xpc.get (target application object)

Property	Description	Writable
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 select the <b>Log Task Execution Time</b> check box located at <b>Simulation menu Configuration Parameters &gt; xPC Target options</b> pane.	No
TimeLog	Storage in the MATLAB workspace for the time or T-vector logged during execution of the target application.	No
ViewMode	Display either all scopes or a single scope on the target computer. Value is 'all' or a single scope index. This property is active only if the environment property TargetScope is set to enabled.	Yes

### Examples

List the value for the target object property StopTime. Notice that the property name is a string, in quotation marks, and not case sensitive.

```
get(tg,'stoptime') or tg.get('stoptime')  
ans = 0.2
```

### See Also

```
get | set | xpctarget.xpc.set (target application object)  
| xpctarget.xpcsc.get (scope object) | xpctarget.xpc.set  
(target application object)
```

**Purpose** All or part of output logs from target object

**Syntax** MATLAB command line

```
log = getlog(target_object, 'log_name', first_point,  
number_samples, decimation)
```

## 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_samples	Number of samples after the start time. This argument is optional. Default is all points in log.
decimation	1 returns all sample points. n returns every nth sample point. This argument is optional. Default is 1.

## Description

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,

```
Out_log = getlog(tg, 'TETLog', 1, 1000)
```

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

```
Output_log = getlog(tg, 'TETLog', 1, 10, 2)
```

```
Time_log = getlog(tg, 'TimeLog', 1, 10, 2)
```

```
plot(Time_log, Output_log)
```

## How To

- `xpctarget.xpc.get` (target application object)
- “Entering Simulation Parameters”

# xpctarget.xpc.getparam

---

**Purpose** Value of target object parameter index

**Syntax** MATLAB command line

```
getparam(target_object, parameter_index)
```

**Arguments**

target_object	Name of a target object. The default name is tg.
parameter_index	Index number of the parameter.

**Description** getparam returns the value of the parameter associated with parameter\_index.

**Examples** Get the value of parameter index 5.

```
getparam(tg, 5)  
ans = 400
```



## Purpose

Parameter index from parameter list

## Syntax

**MATLAB command line**

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

## Arguments

target_object	Name of a target object. The default name is tg.
'block_name'	Simulink block path without model name.
'parameter_name'	Name of a parameter within a Simulink block.

## Description

getparamid 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. Note, enter for block\_name the mangled name that Simulink Coder uses for code generation.

## Examples

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

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

Get the property index of a single block.

```
getparamid(tg, 'Gain1', 'Gain') ans = 5
```

## See Also

xpctarget.xpc.getsignalid

## How To

- “Selected Examples”

## **xpctarget.xpc.getparamid**

---

- “Why Does the getparamid Function Return Nothing?” on page 22-2

**Purpose** Block path and parameter name from index list

**Syntax** MATLAB command line  
`getparamname(target_object, parameter_index)`

**Arguments**

<code>target_object</code>	Name of a target object. The default name is <code>tg</code> .
<code>parameter_index</code>	Index number of the parameter.

**Description** `getparamname` returns two argument strings, block path and parameter name, from the index list for the specified parameter index.

**Examples** Get the block path and parameter name of parameter index 5.

```
[blockPath,parName]=getparamname(tg,5)
blockPath =
Signal Generator
parName =
Amplitude
```

# xpctarget.xpc.getscope

**Purpose** Scope object pointing to scope defined in kernel

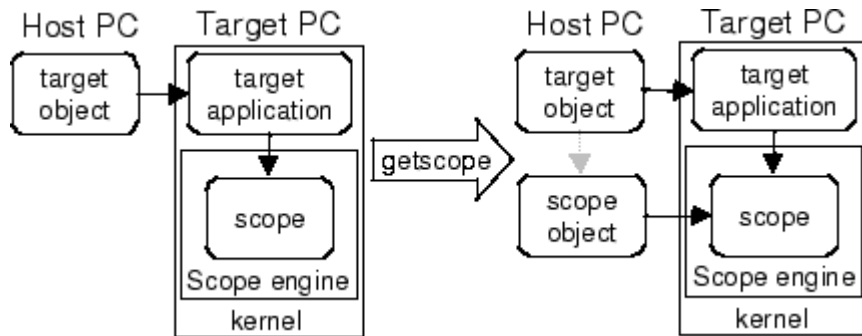
**Syntax** MATLAB command line

```
scope_object_vector = getscope(target_object, scope_number)
scope_object = target_object.getscope(scope_number)
```

**Arguments**

target_object	Name of a target object.
scope_number_vector	Vector of existing scope indices listed in the target object property Scopes. The vector can have only one element.
scope_object	MATLAB variable for a new scope object vector. The vector can have only one scope object.

**Description** getscope returns a scope object vector. If you try to get a nonexistent scope, the result is an error. You can retrieve the list of existing scopes using the method get(target\_object, 'scopes') or target\_object.scopes.



**Examples** If your Simulink model has an xPC Target scope block, a target scope is created at the time the target application is downloaded to the target

computer. 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
```

The following example gets the properties of all scopes on the target computer and creates a vector of scope objects on the host computer. If the target object has more than one scope, it create a vector of scope objects.

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

### See Also

[getxpcenv](#) | [xpctarget.xpc.remscope](#)

### How To

- “Selected Examples”

# xpctarget.xpc.getsignal

---

**Purpose** Value of target object signal index

**Syntax** MATLAB command line  
`getsignal(target_object, signal_index)`

**Arguments**

<code>target_object</code>	Name of a target object. The default name is <code>tg</code> .
<code>signal_index</code>	Index number of the signal. This can be a value of up to 1000 elements.

**Description** `getsignal` returns the value of the signal associated with `signal_index`.

**Examples** Get the value of signal index 2.

```
getsignal(tg, 2)  
ans = -3.3869e+006
```

<b>Purpose</b>	Signal index or signal property from signal list				
<b>Syntax</b>	<b>MATLAB command line</b>  <code>getsignalid(target_object, 'signal_name')</code> <code>tg.getsignalid('signal_name')</code>				
<b>Arguments</b>	<table><tr><td><code>target_object</code></td><td>Name of an existing target object.</td></tr><tr><td><code>signal_name</code></td><td>Enter the name of a signal from your Simulink model. For blocks with a single signal, the <code>signal_name</code> is equal to the <code>block_name</code>. For blocks with multiple signals, the xPC Target software appends S1, S2 ... to the <code>block_name</code>.</td></tr></table>	<code>target_object</code>	Name of an existing target object.	<code>signal_name</code>	Enter the name of a signal from your Simulink model. For blocks with a single signal, the <code>signal_name</code> is equal to the <code>block_name</code> . For blocks with multiple signals, the xPC Target software appends S1, S2 ... to the <code>block_name</code> .
<code>target_object</code>	Name of an existing target object.				
<code>signal_name</code>	Enter the name of a signal from your Simulink model. For blocks with a single signal, the <code>signal_name</code> is equal to the <code>block_name</code> . For blocks with multiple signals, the xPC Target software appends S1, S2 ... to the <code>block_name</code> .				
<b>Description</b>	<code>getsignalid</code> 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. Note, enter for <code>block_name</code> the mangled name that Simulink Coder uses for code generation.				
<b>Examples</b>	Get the signal index for the single signal from the Simulink block Gain1.  <code>getsignalid(tg, 'Gain1')</code> or <code>tg.getsignalid('Gain1')</code> <code>ans = 6</code>				
<b>See Also</b>	<code>xpctarget.xpc.getparamid</code>				
<b>How To</b>	<ul style="list-style-type: none"><li>• “Selected Examples”</li><li>• “Why Does the getparamid Function Return Nothing?” on page 22-2</li></ul>				

# xpctarget.xpc.getsignalidsfromlabel

---

**Purpose** Return vector of signal indices

**Syntax** MATLAB command line

```
getsignalidsfromlabel(target_object, signal_label)  
target_object.getsignalidsfromlabel(signal_label)
```

**Arguments**

target_object	Name of a target object. The default name is tg.
signal_label	Signal label (from Simulink model).

**Description** getsignalidsfromlabel returns a vector of one or more signal indices that are associated with the labeled signal, signal\_label. This function assumes that you have labeled the signal for which you request the index (see the **Signal name** parameter of the “Signal Properties Dialog Box” in the Simulink documentation). Note that the xPC Target software refers to Simulink signal names as signal labels.

**Examples** Get the vector of signal indices for a signal labeled Gain.

```
>> tg.getsignalidsfromlabel('xpcoscGain')  
ans =  
0
```

**See Also** xpctarget.xpc.getsignallabel



**Purpose** Return signal label

**Syntax** MATLAB command line

```
getsignallabel(target_object, signal_index)  
target_object.getsignallabel(signal_index)
```

**Arguments**

target_object	Name of a target object. The default name is tg.
signal_index	Index number of the signal.

**Description** getsignallabel returns the signal label for the specified signal index, signal\_index. signal\_label. This function assumes that you have labeled the signal for which you request the label (see the **Signal name** parameter of the “Signal Properties Dialog Box” in the Simulink documentation). Note that the xPC Target software refers to Simulink signal names as signal labels.

**Examples**

```
>> getsignallabel(tg, 0)  
ans =  
xpcoscGain
```

**See Also** xpctarget.xpc.getsignalidsfromlabel

# xpctarget.xpc.getsignalname

---

**Purpose** Signal name from index list

**Syntax** MATLAB command line

```
getsignalname(target_object, signal_index)  
target_object.getsignalname(signal_index)
```

**Arguments**

target_object	Name of a target object. The default name is tg.
signal_index	Index number of the signal.

**Description** getparamname returns one argument string, signal name, from the index list for the specified signal index.

**Examples** Get the signal name of signal ID 2.

```
[sigName]=getsignalname(tg,2)  
sigName =  
Gain2
```

**Purpose** Determine which PCI boards are installed in target computer

**Syntax** MATLAB command line

```
getxpcpci(target_object, 'type_of_boards')  
getxpcpci(target_object, 'verbose')
```

<b>Arguments</b>	target_object	Variable name to reference the target object.
	type_of_boards	Values are no arguments, 'all', and 'supported'.
	verbose	Argument to include the base address register information in the PCI device display.

## Description

The `getxpcpci` function displays, in the MATLAB window, which PCI boards are installed in the target computer. By default, `getxpcpci` displays this information for the target object, `tg`. If you have multiple target computers in your system, you can call the `getxpcpci` function for a particular target object, `target_object`.

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, base address, and the board PCI ID itself.

The following preconditions must be met before you can use this function:

- The host-target communication link must be working. (The function `xpctargetping` must return `success` before you can use 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 before the model build process.

## Examples

The following example displays the installed PCI devices, not only the devices supported by the xPC Target block library. This includes graphics controllers, network cards, SCSI cards, and even devices that are part of the motherboard chip set (for example, PCI-to-PCI bridges).

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

The following example displays a list of the currently supported PCI devices in the xPC Target block library, including subvendor and subdevice information.

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

The following example displays a list of the currently supported PCI devices in the xPC Target block library, including subvendor and subdevice information and base address register contents.

```
getxpcpci('verbose')
```

When called with the 'supported' option, `getxpcpci` does not access the target computer.

To display the list of PCI devices installed on the target computer, `tg1`, first create a target object, `tg1`, for that target computer. Then, call `getxpcpci` with the 'all' option. For example:

```
tg1=xpctarget.xpc('RS232','COM1','115200')
getxpcpci(tg1, 'all')
```

To return the result of a `getxpcpci` query in the struct `pcidevs` instead of displaying it, assign the function to `pcidevs`. The struct `pcidevs` is an array with one element for each detected PCI device. Each element combines the information by a set of field names. The struct contains more information compared to the displayed list. Its contents vary according to the options you specify for the function.

```
pcidevs = getxpcpci
```

# xpctarget.xpc.load

---

**Purpose** Download target application to target computer

**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** Before using this function, the target computer must be booted with the xPC Target kernel, and the target application must be built in the current working folder on the host computer.

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 Simulink Coder build process.

---

**Note** If you are running in Standalone mode, this command has no effect. To load a new application, you must rebuild the standalone application with the new application, then reboot the target computer with the updated standalone application.

---

**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** xpctarget.xpc.unload

**How To** • “Selected Examples”

**Purpose** Restore parameter values saved in specified file

**Syntax** MATLAB command line

```
loadparamset(target_object,'filename')  
target_object.loadparamset('filename')
```

**Arguments**

target_object	Name of an existing target object.
filename	Enter the name of the file that contains the saved parameters.

**Description**

loadparamset restores the target application parameter values saved in the file filename. This file must be located on a local drive of the target computer. This method assumes that you have a parameter file from a previous run of the xpctarget.xpc.saveparamset method.

**See Also**

xpctarget.xpc.saveparamset

# xpctarget.xpc.reboot

---

**Purpose** Reboot target computer

**Syntax** MATLAB command line

`reboot(target_object)`

**Target computer command line**

`reboot`

**Arguments** `target_object` Name of an existing target object.

**Description** `reboot` reboots the target computer, and if a target boot disk is still present, the xPC Target kernel is reloaded.

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

---

**Note** This method might not work on some target hardware.

---

**See Also** `xpctarget.xpc.load` | `xpctarget.xpc.unload`



**Purpose** Remove scope from target computer

**Syntax** MATLAB command line

```
remscope(target_object, scope_number_vector)
target_object.remscope(scope_number_vector)
remscope(target_object)
target_object.remscope
```

**Target computer command line**

```
remscope scope_number
remscope 'all'
```

## Arguments

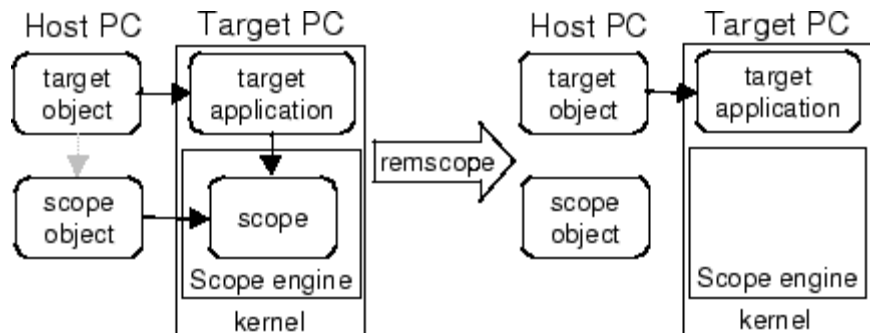
**target\_object** Name of a target object. The default name is tg.

**scope\_number\_vector** Vector of existing scope indices listed in the target object property Scopes.

**scope\_number** Single scope index.

## Description

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



Note that you can only permanently remove scopes that are added with the method `addscope`. This is a scope that is outside a model. If you remove a scope that has been added through a scope block (the scope block is inside the model), a subsequent run of that model creates the scope again.

## 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

`xpctarget.xpc.addscope` | `xpctarget.xpc.getscope`

## How To

- “Selected Examples”

**Purpose** Save current target application parameter values

**Syntax** MATLAB command line

```
saveparamset(target_object,'filename')  
target_object.saveparamset('filename')
```

**Arguments**

target_object	Name of an existing target object.
filename	Enter the name of the file to contain the saved parameters.

**Description**

saveparamset saves the target application parameter values in the file filename. This method saves the file on a local drive of the target computer (C:\ by default). You can later reload these parameters with the xpctarget.xpc.loadparamset function.

You might want to save target application parameter values if you change these parameter values while the application is running in real time. Saving these values enables you to easily recreate target application parameter values from a number of application runs.

**See Also**

xpctarget.xpc.loadparamset

# xpctarget.xpc.set (target application object)

---

**Purpose** Change target application object property values

**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.property_name = property_value
```

**Target computer 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 target object property. Always use quotation marks.
property_value	Value for a target object property. Always use quotation marks for character strings; quotation marks are optional for numbers.

**Description**

set 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 must 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 writable properties for a target object

## xpctarget.xpc.set (target application object)

---

are listed in the following table. This table includes a description of the properties:

Property	Description	Writable
CommunicationTimeout	Communication timeout between host and target computer, in seconds.	Yes
LogMode	Controls which data points are logged: <ul style="list-style-type: none"><li>• Time-equidistant logging. Logs a data point at every time interval. Set value to 'Normal'.</li><li>• Value-equidistant logging. Logs a data point only when an output signal from the OutputLog changes by a specified value (increment). Set the value to the difference in signal values.</li></ul>	Yes
SampleTime	Time between samples. This value equals the step size, in seconds, for updating the model equations and posting the outputs. See “User Interaction” in the <i>xPC Target Getting Started Guide</i> for limitations on target property changes to sample times.	Yes

## xpctarget.xpc.set (target application object)

---

Property	Description	Writable
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' and 'off'.	Yes
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' and 'off'.	Yes
StopTime	Time when the target application stops running. Values are in seconds. The original value is set in the <b>Simulation</b> menu <b>Configuration Parameters</b> dialog.  When the ExecTime reaches StopTime, the application stops running.	Yes
ViewMode	Display either all scopes or a single scope on the target computer. Value is 'all' or a single scope index. This property is active only if the environment property TargetScope is set to enabled.	Yes

# xpctarget.xpc.set (target application object)

---

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.

```
set(tg)
ans =
    StopTime: {}
    SampleTime: {}
    ViewMode: {}
    LogMode: {}
    ShowParameters: {}
    ShowSignals: {}
```

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

`get` | `set` | `xpctarget.xpc.get` (target application object) | `xpctarget.xpcsc.get` (scope object) | `xpctarget.xpcsc.set` (scope object)

## How To

- “Selected Examples”

# xpctarget.xpc.setparam

---

**Purpose** Change writable target object parameters

**Syntax** MATLAB command line

```
setparam(target_object, parameter_index, parameter_value)
```

## Arguments

target_object	Name of an existing target object. The default name is tg.
parameter_index	Index number of the parameter.
parameter_value	Value for a target object parameter.

## Description

Method of a target object. Set the value of the target parameter. This method returns a structure that stores the parameter index, previous parameter values, and new parameter values in the following fields:

- parIndexVec
- OldValues
- NewValues

## Examples

Set the value of parameter index 5 to 100.

```
setparam(tg, 5, 100)
ans =
parIndexVec: 5
OldValues: 400
NewValues: 100
```

Simultaneously set values for multiple parameters. Use the cell array format to specify new parameter values.

```
setparam(tg, [1 5], {10,100})
ans =
parIndexVec: [1 5]
OldValues: {[2] [4]}
```



NewValues: {[10] [100]}

# xpctarget.xpc.start (target application object)

---

**Purpose** Start execution of target application on target computer

**Syntax** MATLAB command line

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

**Target computer 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 computer. If a target application is running, this command has no effect.

**Examples** Start the target application represented by the target object tg.

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

**See Also** xpctarget.xpc.stop (target application object)  
| xpctarget.xpc.load | xpctarget.xpc.unload |  
xpctarget.xpcsc.stop (scope object)

# xpctarget.xpc.stop (target application object)

---

**Purpose** Stop execution of target application on target computer

**Syntax** MATLAB command line

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

**Target computer 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** xpctarget.xpc.start (target application object) | xpctarget.xpcsc.stop (scope object) | xpctarget.xpcsc.start (scope object)

# xpctarget.xpc.targetping

---

**Purpose** Test communication between host and target computers

**Syntax** MATLAB command line  
  
targetping(target\_object)  
target\_object.targetping

**Arguments** target\_object Name of a target object.

**Description** Method of a target object. Use this method to ping a target computer from the host computer. This method returns `success` if the xPC Target kernel is loaded and running and communication is working between host and target, otherwise it returns `failed`.

This function works with both RS-232 and TCP/IP communication.

---

**Note** RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.

---

**Examples** Ping the communication between the host and the target object tg.  
  
targetping(tg) or tg.targetping

**See Also** xpctarget.xpc

**Purpose** Remove current target application from target computer

**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 and is ready to download new target application from the host computer.

---

**Note** If you are running in StandAlone mode, this command has no effect. To unload and reload a new application, you must rebuild the standalone application with the new application, then reboot the target computer with the updated standalone application.

---

**Examples** Unload the target application represented by the target object tg.

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

**See Also** xpctarget.xpc.load | xpctarget.xpc.reboot

# xpctarget.xpcfs Class

**Purpose** Control and access properties of file scopes

**Description** The scope gets a data package from the kernel and stores the data in a file in the target computer file system. Depending on the setting of WriteMode, the file size is or is not continuously updated. You can then transfer the data to another computer for examination or plotting.

## Methods

These methods are inherited from xpctarget.xpcsc Class.

Method	Description
xpctarget.xpcsc.addsignals	Add signals to scope represented by scope object
xpctarget.xpcsc.get(scope object)	Return property values for scope objects
xpctarget.xpcsc.removesignals	Remove signals from scope represented by scope object
xpctarget.xpcsc.set(scope object)	Change property values for scope objects
xpctarget.xpcsc.start(scope object)	Start execution of scope on target computer
xpctarget.xpcsc.stop(scope object)	Stop execution of scope on target computer
xpctarget.xpcsc.trigger	Software-trigger start of data acquisition for scope(s)

## Properties

These properties are inherited from xpctarget.xpcsc Class.

Property	Description	Writable
Application	Name of the Simulink model associated with this scope object.	No
Decimation	A number n, where every nth sample is acquired in a scope window.	Yes

Property	Description	Writable
NumPrePostSamples	For host or target scopes, this parameter is the 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 <b>TriggerMode</b> to 'FreeRun', this property has no effect on data acquisition.	Yes
NumSamples	<p>Number of contiguous samples captured during the acquisition of a data package. If the scope stops before capturing this number of samples, the scope has the collected data up to the end of data collection, then has zeroes for the remaining uncollected data. Note that you should know what type of data you are collecting, it is possible that your data contains zeroes.</p> <p>For file scopes, this parameter works in conjunction with the <b>AutoRestart</b> check box. If the <b>AutoRestart</b> box is selected, the file scope collects data up to <b>Number of Samples</b>, then starts over again, overwriting the buffer. If the <b>AutoRestart</b> box is not selected, the file scope collects data only up to <b>Number of Samples</b>, then stops.</p>	Yes
ScopeId	A numeric index, unique for each scope.	No
Signals	List of signal indices from the target object to display on the scope.	Yes
Status	Indicate 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'.	No

## xpctarget.xpcfs Class

Property	Description	Writable
TriggerLevel	If TriggerMode is '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
TriggerMode	Trigger mode for a scope. Valid values are 'FreeRun' (default), 'Software', 'Signal', and 'Scope'.	Yes
TriggerSample	<p>If TriggerMode is 'Scope', then TriggerSample specifies which sample of the triggering scope the current scope should trigger on. For example, if TriggerSample is 0 (default), the current scope triggers on sample 0 (first sample acquired) of the triggering scope. This means that the two scopes will be perfectly synchronized. If TriggerSample is 1, the first sample (sample 0) of the current scope will be at the same instant as sample number 1 (second sample in the acquisition cycle) of the triggering scope.</p> <p>As a special case, setting TriggerSample to -1 means that the current scope is triggered at the end of the acquisition cycle of the triggering scope. Thus, the first sample of the triggering scope is acquired one sample after the last sample of the triggering scope.</p>	Yes
TriggerScope	If TriggerMode is 'Scope', identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered. You do this by setting the slave scope property TriggerScope to the scope index of the master scope.	Yes



Property	Description	Writable
TriggerSignal	If TriggerMode is 'Signal', identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property Signal.	Yes
TriggerSlope	If TriggerMode is 'Signal', indicates whether the trigger is on a rising or falling signal. Values are 'Either' (default), 'Rising', and 'Falling'.	Yes
Type	Determines whether the scope is displayed on the host computer or on the target computer. Values are 'Host', 'Target', and 'File'.	Yes

These properties are specific to class xpcfs.

Property	Description	Writeable
AutoRestart	<p>Values are 'on' and 'off'.</p> <p>For file scopes, enable the file scope to collect data up to the number of samples (NumSamples), then start over again, appending the new data to the end of the signal data file. Clear the <b>AutoRestart</b> check box to have the file scope collect data up to <b>Number of samples</b>, then stop.</p> <p>If the named signal data file already exists when you start the target application, the software overwrites the old data with the new signal data.</p> <p>To use the DynamicFileName property, set AutoRestart to 'on' first.</p>	No

# xpctarget.xpcf Class

Property	Description	Writeable
	<p>For host or target scopes, this parameter has no effect.</p>	
DynamicFileName	<p>Values are 'on' and 'off'. By default, the value is 'off'.</p> <p>Enable the ability to dynamically create multiple log files for file scopes.</p> <p>To use DynamicFileName, set AutoRestart to 'on' first. When you enable DynamicFileName, configure Filename to create incrementally numbered file names for the multiple log files. Failure to do so causes an error when you try to start the scope.</p> <p>You can enable the creation of up to 99999999 files (&lt;%%%%%%%%&gt;.dat). The length of a file name, including the specifier, cannot exceed eight characters.</p> <p>For host or target scopes, this parameter has no effect.</p>	Yes

Property	Description	Writeable
Filename	<p>Provide a name for the file to contain the signal data. By default, the target computer writes the signal data to a file named C:\data.dat for scope blocks. Note that for file scopes created through the MATLAB interface, there is no name initially assigned to <code>FileName</code>. After you start the scope, the software assigns a name for the file to acquire the signal data. This name typically consists of the scope object name, <code>ScopeId</code>, and the beginning letters of the first signal added to the scope.</p> <p>If you set <code>DynamicFileName</code> and <code>AutoRestart</code> to 'on', configure <code>Filename</code> to dynamically increment. Use a base file name, an underscore (<code>_</code>), and a <code>&lt; &gt;</code> specifier. Within the specifier, enter one to eight <code>%</code> symbols. Each symbol <code>%</code> represents a decimal location in the file name. The specifier can appear anywhere in the file name. For example, the following value for <code>Filename</code>, <code>C:\work\file_&lt;%%&gt;.dat</code> creates file names with the following pattern:</p> <pre>file_001.dat file_002.dat file_003.dat</pre>	No

# xpctarget.xpcfs Class

Property	Description	Writeable
	<p>The last file name of this series will be <code>file_999.dat</code>. If the function is still logging data when the last file name reaches its maximum size, the function starts from the beginning and overwrites the first file name in the series. If you do not retrieve the data from existing files before they are overwritten, the data is lost.</p> <p>For host or target scopes, this parameter has no effect.</p>	
<code>MaxWriteFileSize</code>	<p>Provide the maximum size of <code>Filename</code>, in bytes. This value must be a multiple of <code>WriteSize</code>. Default is <code>536870912</code>.</p> <p>When the size of a log file reaches <code>MaxWriteFileSize</code>, the software creates a subsequently numbered file name, and continues logging data to that file, up until the highest log file number you have specified. If the software cannot create any additional log files, it overwrites the first log file.</p> <p>For host or target scopes, this parameter has no effect.</p>	Yes

Property	Description	Writeable
Mode	<p><b>Note</b> The Mode property will be removed in a future release.</p> <ul style="list-style-type: none"><li>• For target scopes, use <code>DisplayMode</code>.</li><li>• For file scopes, use <code>WriteMode</code>.</li><li>• For host scopes, this parameter has no effect.</li></ul>	Yes
WriteMode	<p>For file scopes, specify when a file allocation table (FAT) entry is updated. Values are 'Lazy' or 'Commit'. Both modes write the signal data to the file. With 'Commit' mode, each file write operation simultaneously updates the FAT entry for the file. This mode is slower, but the file system always knows the actual file size. With 'Lazy' mode, the FAT entry is updated only when the file is closed and not during each file write operation. This mode is faster, but if the system crashes before the file is closed, the file system might not know the actual file size (the file contents, however, will be intact).</p>	Yes

## xpctarget.xpcfs Class

---

Property	Description	Writeable
	For host or target scopes, this parameter has no effect.	
WriteSize	<p>Enter the block size, in bytes, of the data chunks. This parameter specifies that a memory buffer, of length number of samples (NumSamples), collect data in multiples of WriteSize. By default, this parameter is 512 bytes, which is the typical disk sector size. Using a block size that is the same as the disk sector size provides optimal performance.</p> <p>If you experience a system crash, you can expect to lose an amount of data the size of WriteSize.</p> <p>For host or target scopes, this parameter has no effect.</p>	Yes

**Purpose** Add signals to scope represented by 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

`addsignal` adds signals to a scope object. The signals must be specified by their indices, which you can 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.

---

**Note** You must stop the scope before you can add a signal to it.

---

## 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])
```

## xpctarget.xpcsc.addsignal

---

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

```
sc1.Signals
Signals          = 1 : Signal Generator
                  0 : Integrator1
```

Another way 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

`xpctarget.xpcsc.remsignal` | `xpctarget.xpcsc.set` (scope object) | `xpctarget.xpc.addscope` | `xpctarget.xpc.getsignalid`



# xpctarget.xpcsc.get (scope object)

**Purpose** Return 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

`get` 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. 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 descriptions of the properties and the properties you can change directly by assigning a value.

Property	Description	Writable
Application	Name of the Simulink model associated with this scope object.	No
AutoRestart	Values are 'on' and 'off'.  For file scopes, enable the file scope to collect data up to the number of samples ( <b>NumSamples</b> ), then start over again, appending the new data to the end of the signal data file. Clear the <b>AutoRestart</b> check box to have the file scope collect data up to <b>Number of samples</b> , then stop.	No

## xpctarget.xpcsc.get (scope object)

Property	Description	Writable
	<p>If the named signal data file already exists when you start the target application, the software overwrites the old data with the new signal data.</p> <p>For host or target scopes, this parameter has no effect.</p> <p>To use the <code>DynamicFileName</code> property, set <code>AutoRestart</code> to 'on' first.</p>	
Data	<p>Contains the output data for a single data package from a scope.</p> <p>For target or file scopes, this parameter has no effect.</p>	No
Decimation	<p>A number <math>n</math>, where every <math>n</math>th sample is acquired in a scope window.</p>	Yes
DisplayMode	<p>For target scopes, indicate how a scope displays the signals. Values are 'Numerical', 'Redraw' (default), 'Sliding', and 'Rolling'.</p> <p>For host or file scopes, this parameter has no effect.</p> <p>.</p>	Yes
DynamicFileName	<p>Values are 'on' and 'off'. By default, the value is 'off'.</p> <p>Enable the ability to dynamically create multiple log files for file scopes.</p> <p>To use <code>DynamicFileName</code>, set <code>AutoRestart</code> to 'on' first. When you enable <code>DynamicFileName</code>, configure <code>Filename</code> to create incrementally numbered file names for the multiple log files. Failure to do so causes an error when you try to start the scope.</p>	Yes

## xpctarget.xpcsc.get (scope object)

Property	Description	Writable
	<p>You can enable the creation of up to 99999999 files (&lt;%%%%%%%%&gt;.dat). The length of a file name, including the specifier, cannot exceed eight characters.</p> <p>For host or file scopes, this parameter has no effect.</p>	
Filename	<p>Provide a name for the file to contain the signal data. By default, the target computer writes the signal data to a file named C:\data.dat for scope blocks. Note that for file scopes created through the MATLAB interface, there is no name initially assigned to FileName. After you start the scope, the software assigns a name for the file to acquire the signal data. This name typically consists of the scope object name, ScopeId, and the beginning letters of the first signal added to the scope.</p> <p>If you set DynamicFileName and AutoRestart to 'on', configure Filename to dynamically increment. Use a base file name, an underscore (_), and a &lt; &gt; specifier. Within the specifier, enter one to eight % symbols. Each symbol % represents a decimal location in the file name. The specifier can appear anywhere in the file name. For example, the following value for Filename, C:\work\file_&lt;%%&gt;.dat creates file names with the following pattern:</p> <pre>file_001.dat file_002.dat file_003.dat</pre> <p>The last file name of this series will be file_999.dat. If the function is still logging data when the last file name reaches its maximum</p>	No

## xpctarget.xpcsc.get (scope object)

Property	Description	Writable
	<p>size, the function starts from the beginning and overwrites the first file name in the series. If you do not retrieve the data from existing files before they are overwritten, the data is lost.</p> <p>For host or target scopes, this parameter has no effect.</p>	
MaxWriteFileSize	<p>Provide the maximum size of Filename, in bytes. This value must be a multiple of WriteSize. Default is 536870912.</p> <p>When the size of a log file reaches MaxWriteFileSize, the software creates a subsequently numbered file name, and continues logging data to that file, up until the highest log file number you have specified. If the software cannot create any additional log files, it overwrites the first log file.</p>	Yes
Grid	<p>Values are 'on' and 'off'.</p> <p>For host or file scopes, this parameter has no effect.</p>	Yes
Mode	<hr/> <p><b>Note</b> The Mode property will be removed in a future release.</p> <ul style="list-style-type: none"> <li>• For target scopes, use DisplayMode.</li> <li>• For file scopes, use WriteMode.</li> <li>• For host scopes, this parameter has no effect.</li> </ul> <hr/>	Yes

## xpctarget.xpcsc.get (scope object)

Property	Description	Writable
NumPrePostSamples	For host or target scopes, this parameter is the 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 <b>TriggerMode</b> to 'FreeRun', this property has no effect on data acquisition.	Yes
NumSamples	<p>Number of contiguous samples captured during the acquisition of a data package. If the scope stops before capturing this number of samples, the scope has the collected data up to the end of data collection, then has zeroes for the remaining uncollected data. Note that you should know what type of data you are collecting, it is possible that your data contains zeroes.</p> <p>For file scopes, this parameter works in conjunction with the <b>AutoRestart</b> check box. If the <b>AutoRestart</b> box is selected, the file scope collects data up to <b>Number of Samples</b>, then starts over again, overwriting the buffer. If the <b>AutoRestart</b> box is not selected, the file scope collects data only up to <b>Number of Samples</b>, then stops.</p>	Yes
ScopeId	A numeric index, unique for each scope.	No
Signals	List of signal indices from the target object to display on the scope.	Yes
Status	Indicate 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'.	No

## xpctarget.xpcsc.get (scope object)

Property	Description	Writable
Time	Contains the time data for a single data package from a scope.	No
TriggerLevel	If TriggerMode is '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
TriggerMode	Trigger mode for a scope. Valid values are 'FreeRun' (default), 'Software', 'Signal', and 'Scope'.	Yes
TriggerSample	<p>If TriggerMode is 'Scope', then TriggerSample specifies which sample of the triggering scope the current scope should trigger on. For example, if TriggerSample is 0 (default), the current scope triggers on sample 0 (first sample acquired) of the triggering scope. This means that the two scopes will be perfectly synchronized. If TriggerSample is 1, the first sample (sample 0) of the current scope will be at the same instant as sample number 1 (second sample in the acquisition cycle) of the triggering scope.</p> <p>As a special case, setting TriggerSample to -1 means that the current scope is triggered at the end of the acquisition cycle of the triggering scope. Thus, the first sample of the triggering scope is acquired one sample after the last sample of the triggering scope.</p>	Yes
TriggerScope	If TriggerMode is 'Scope', identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered. You do this by setting the slave scope property TriggerScope to the scope index of the master scope.	Yes

## xpctarget.xpcsc.get (scope object)

Property	Description	Writable
TriggerSignal	If TriggerMode is 'Signal', identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property Signal.	Yes
TriggerSlope	If TriggerMode is 'Signal', indicates whether the trigger is on a rising or falling signal. Values are 'Either' (default), 'Rising', and 'Falling'.	Yes
Type	Determines whether the scope is displayed on the host computer or on the target computer. Values are 'Host', 'Target', and 'File'.	Yes
WriteMode	<p>For file scopes, specify when a file allocation table (FAT) entry is updated. Values are 'Lazy' or 'Commit'. Both modes write the signal data to the file. With 'Commit' mode, each file write operation simultaneously updates the FAT entry for the file. This mode is slower, but the file system always knows the actual file size. With 'Lazy' mode, the FAT entry is updated only when the file is closed and not during each file write operation. This mode is faster, but if the system crashes before the file is closed, the file system might not know the actual file size (the file contents, however, will be intact).</p> <p>For host or target scopes, this parameter has no effect.</p>	Yes

## xpctarget.xpcsc.get (scope object)

Property	Description	Writable
WriteSize	<p>Enter the block size, in bytes, of the data chunks. This parameter specifies that a memory buffer, of length number of samples (NumSamples), collect data in multiples of WriteSize. By default, this parameter is 512 bytes, which is the typical disk sector size. Using a block size that is the same as the disk sector size provides optimal performance.</p> <p>If you experience a system crash, you can expect to lose an amount of data the size of WriteSize.</p> <p>For host or target scopes, this parameter has no effect.</p>	Yes
YLimit	<p>Minimum and maximum y-axis values. This property can be set to 'auto'.</p> <p>For host or file scopes, this parameter has no effect.</p>	Yes

### Examples

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

```
get(sc)
```

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

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

### See Also

```
get | set | xpctarget.xpcsc.set (scope object) |  
xpctarget.xpc.set (target application object)
```



**Purpose** Base class for all scope classes

**Description** This is the base class for the scope classes, `xpctarget.xpcfs Class`, `xpctarget.xpcschost Class`, and `xpctarget.xpcscctg Class`. All methods and properties are inherited by the derived classes. When a mixture of derived classes are stored in a scope collection, only the base class methods and properties are available. All scope class constructors are `Private` and are not intended to be called from the MATLAB prompt.

A scope acquires data from the target application and displays that data on the target computer, uploads the data to the host computer, or stores that data in a file in the target computer file system. All target, host, or file scopes run on the target computer.

### Methods

These methods are inherited by the derived classes.

Method	Description
<code>xpctarget.xpcsc.addsignals</code>	Add signals to scope represented by scope object
<code>xpctarget.xpcsc.get</code> (scope object)	Return property values for scope objects
<code>xpctarget.xpcsc.removesignals</code>	Remove signals from scope represented by scope object
<code>xpctarget.xpcsc.set</code> (scope object)	Change property values for scope objects
<code>xpctarget.xpcsc.start</code> (scope object)	Start execution of scope on target computer
<code>xpctarget.xpcsc.stop</code> (scope object)	Stop execution of scope on target computer
<code>xpctarget.xpcsc.trigger</code>	Software trigger start of data acquisition for scope(s)

### Properties

These properties are inherited by the derived classes.

## xpctarget.xpcsc Class

Property	Description	Writable
Application	Name of the Simulink model associated with this scope object.	No
Decimation	A number n, where every nth sample is acquired in a scope window.	Yes
NumPrePostSamples	For host or target scopes, this parameter is the 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 <code>TriggerMode</code> to 'FreeRun', this property has no effect on data acquisition.	Yes
NumSamples	<p>Number of contiguous samples captured during the acquisition of a data package. If the scope stops before capturing this number of samples, the scope has the collected data up to the end of data collection, then has zeroes for the remaining uncollected data. Note that you should know what type of data you are collecting, it is possible that your data contains zeroes.</p> <p>For file scopes, this parameter works in conjunction with the <b>AutoRestart</b> check box. If the <b>AutoRestart</b> box is selected, the file scope collects data up to <b>Number of Samples</b>, then starts over again, overwriting the buffer. If the <b>AutoRestart</b> box is not selected, the file scope collects data only up to <b>Number of Samples</b>, then stops.</p>	Yes
ScopeId	A numeric index, unique for each scope.	No
Signals	List of signal indices from the target object to display on the scope.	Yes

Property	Description	Writable
Status	Indicate 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'.	No
TriggerLevel	If TriggerMode is '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
TriggerMode	Trigger mode for a scope. Valid values are 'FreeRun' (default), 'Software', 'Signal', and 'Scope'.	Yes
TriggerSample	<p>If TriggerMode is 'Scope', then TriggerSample specifies which sample of the triggering scope the current scope should trigger on. For example, if TriggerSample is 0 (default), the current scope triggers on sample 0 (first sample acquired) of the triggering scope. This means that the two scopes will be perfectly synchronized. If TriggerSample is 1, the first sample (sample 0) of the current scope will be at the same instant as sample number 1 (second sample in the acquisition cycle) of the triggering scope.</p> <p>As a special case, setting TriggerSample to -1 means that the current scope is triggered at the end of the acquisition cycle of the triggering scope. Thus, the first sample of the triggering scope is acquired one sample after the last sample of the triggering scope.</p>	Yes

## xpctarget.xpcsc Class

---

<b>Property</b>	<b>Description</b>	<b>Writable</b>
TriggerScope	If TriggerMode is 'Scope', identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered. You do this by setting the slave scope property TriggerScope to the scope index of the master scope.	Yes
TriggerSignal	If TriggerMode is 'Signal', identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property Signal.	Yes
TriggerSlope	If TriggerMode is 'Signal', indicates whether the trigger is on a rising or falling signal. Values are 'Either' (default), 'Rising', and 'Falling'.	Yes
Type	Determines whether the scope is displayed on the host computer or on the target computer. Values are 'Host', 'Target', and 'File'.	Yes

**Purpose** Remove signals from scope represented by 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 method <code>addscope</code> or <code>getscope</code> .
signal_index_vector	Index numbers from the scope object property <code>Signals</code> . This argument is optional, and if it is left out all signals are removed.
signal_index	Single signal index.

**Description** `remsignal` removes signals from a scope object. The signals must be specified by their indices, which you can retrieve using the target object method `getsignalid`. If the `scope_index_vector` has two or more scope objects, the same signals are removed from each scope. The argument `signal_index` is optional; if it is left out, all signals are removed.

---

**Note** You must stop the scope before you can remove a signal from it.

---

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

```
sc1.get('signals')
ans= 0 1
```

# xpctarget.xpcsc.remsignal

---

Remove signals from the scope on the target computer with the scope object property `Signals` updated.

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

or

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

## See Also

[xpctarget.xpcsc.remsignal](#) | [xpctarget.xpc.getsignalid](#)

**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_value, . . .)
```

**Arguments**

<code>scope_object</code>	Name of a scope object or a vector of scope objects.
<code>'property_name'</code>	Name of a scope object property. Always use quotation marks.
<code>property_value</code>	Value for a scope object property. Always use quotation marks for character strings; quotation marks are optional for numbers.

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

Properties must be entered in pairs or, using the alternate syntax, as one-dimensional cell arrays of the same size. This means they must 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 values of the properties after the indicated settings have been made.

## xpctarget.xpcsc.set (scope object)

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

Property	Description	Writable
Application	Name of the Simulink model associated with this scope object.	No
AutoRestart	Values are 'on' and 'off'.  For file scopes, enable the file scope to collect data up to the number of samples (NumSamples), then start over again, appending the new data to the end of the signal data file. Clear the <b>AutoRestart</b> check box to have the file scope collect data up to <b>Number of samples</b> , then stop.  If the named signal data file already exists when you start the target application, the software overwrites the old data with the new signal data.  For host or target scopes, this parameter has no effect.  To use the DynamicFileName property, set AutoRestart to 'on' first.	No
Data	Contains the output data for a single data package from a scope.  For target or file scopes, this parameter has no effect.	No
Decimation	A number n, where every nth sample is acquired in a scope window.	Yes



## xpctarget.xpcsc.set (scope object)

Property	Description	Writable
DisplayMode	<p>For target scopes, indicate how a scope displays the signals. Values are 'Numerical', 'Redraw' (default), 'Sliding', and 'Rolling'.</p> <p>For host or file scopes, this parameter has no effect.</p> <p>.</p>	Yes
DynamicFileName	<p>Values are 'on' and 'off'. By default, the value is 'off'.</p> <p>Enable the ability to dynamically create multiple log files for file scopes.</p> <p>To use DynamicFileName, set AutoRestart to 'on' first. When you enable DynamicFileName, configure Filename to create incrementally numbered file names for the multiple log files. Failure to do so causes an error when you try to start the scope.</p> <p>You can enable the creation of up to 99999999 files (&lt;%%%%%%%%&gt;.dat). The length of a file name, including the specifier, cannot exceed eight characters.</p> <p>For host or file scopes, this parameter has no effect.</p>	Yes

## xpctarget.xpcsc.set (scope object)

Property	Description	Writable
Filename	<p>Provide a name for the file to contain the signal data. By default, the target computer writes the signal data to a file named C:\data.dat for scope blocks. Note that for file scopes created through the MATLAB interface, there is no name initially assigned to FileName. After you start the scope, the software assigns a name for the file to acquire the signal data. This name typically consists of the scope object name, ScopeId, and the beginning letters of the first signal added to the scope.</p> <p>If you set DynamicFileName and AutoRestart to 'on', configure Filename to dynamically increment. Use a base file name, an underscore ( _ ), and a &lt; &gt; specifier. Within the specifier, enter one to eight % symbols. Each symbol % represents a decimal location in the file name. The specifier can appear anywhere in the file name. For example, the following value for Filename, C:\work\file_&lt;%%&gt;.dat creates file names with the following pattern:</p> <pre>file_001.dat file_002.dat file_003.dat</pre> <p>The last file name of this series will be file_999.dat. If the function is still logging data when the last file name reaches its maximum size, the function starts from the beginning and overwrites the first file name in the series. If you do not retrieve the data from existing files before they are overwritten, the data is lost.</p> <p>For host or target scopes, this parameter has no effect.</p>	No

## xpctarget.xpcsc.set (scope object)

Property	Description	Writable
MaxWriteFileSize	<p>Provide the maximum size of Filename, in bytes. This value must be a multiple of WriteSize. Default is 536870912.</p> <p>When the size of a log file reaches MaxWriteFileSize, the software creates a subsequently numbered file name, and continues logging data to that file, up until the highest log file number you have specified. If the software cannot create any additional log files, it overwrites the first log file.</p>	Yes
Grid	<p>Values are 'on' and 'off'.</p> <p>For host or file scopes, this parameter has no effect.</p>	Yes
Mode	<hr/> <p><b>Note</b> The Mode property will be removed in a future release.</p> <ul style="list-style-type: none"><li>• For target scopes, use DisplayMode.</li><li>• For file scopes, use WriteMode.</li><li>• For host scopes, this parameter has no effect.</li></ul> <hr/>	Yes
NumPrePostSamples	<p>For host or target scopes, this parameter is the 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.</p>	Yes

## xpctarget.xpcsc.set (scope object)

Property	Description	Writable
NumSamples	<p>Number of contiguous samples captured during the acquisition of a data package. If the scope stops before capturing this number of samples, the scope has the collected data up to the end of data collection, then has zeroes for the remaining uncollected data. Note that you should know what type of data you are collecting, it is possible that your data contains zeroes.</p> <p>For file scopes, this parameter works in conjunction with the <b>AutoRestart</b> check box. If the <b>AutoRestart</b> box is selected, the file scope collects data up to <b>Number of Samples</b>, then starts over again, overwriting the buffer. If the <b>AutoRestart</b> box is not selected, the file scope collects data only up to <b>Number of Samples</b>, then stops.</p>	Yes
ScopeId	A numeric index, unique for each scope.	No
Signals	List of signal indices from the target object to display on the scope.	Yes
Status	Indicate 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'.	No
Time	Contains the time data for a single data package from a scope.	No
TriggerLevel	If TriggerMode is '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

## xpctarget.xpcsc.set (scope object)

Property	Description	Writable
TriggerMode	Trigger mode for a scope. Valid values are 'FreeRun' (default), 'Software', 'Signal', and 'Scope'.	Yes
TriggerSample	<p>If TriggerMode is 'Scope', then TriggerSample specifies which sample of the triggering scope the current scope should trigger on. For example, if TriggerSample is 0 (default), the current scope triggers on sample 0 (first sample acquired) of the triggering scope. This means that the two scopes will be perfectly synchronized. If TriggerSample is 1, the first sample (sample 0) of the current scope will be at the same instant as sample number 1 (second sample in the acquisition cycle) of the triggering scope.</p> <p>As a special case, setting TriggerSample to -1 means that the current scope is triggered at the end of the acquisition cycle of the triggering scope. Thus, the first sample of the triggering scope is acquired one sample after the last sample of the triggering scope.</p>	Yes
TriggerScope	If TriggerMode is 'Scope', identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered. You do this by setting the slave scope property TriggerScope to the scope index of the master scope.	Yes
TriggerSignal	If TriggerMode is 'Signal', identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property Signal.	Yes
TriggerSlope	If TriggerMode is 'Signal', indicates whether the trigger is on a rising or falling signal. Values are 'Either' (default), 'Rising', and 'Falling'.	Yes

## xpctarget.xpcsc.set (scope object)

Property	Description	Writable
Type	Determines whether the scope is displayed on the host computer or on the target computer. Values are 'Host', 'Target', and 'File'.	Yes
WriteMode	<p>For file scopes, specify when a file allocation table (FAT) entry is updated. Values are 'Lazy' or 'Commit'. Both modes write the signal data to the file. With 'Commit' mode, each file write operation simultaneously updates the FAT entry for the file. This mode is slower, but the file system always knows the actual file size. With 'Lazy' mode, the FAT entry is updated only when the file is closed and not during each file write operation. This mode is faster, but if the system crashes before the file is closed, the file system might not know the actual file size (the file contents, however, will be intact).</p> <p>For host or target scopes, this parameter has no effect.</p>	Yes
WriteSize	<p>Enter the block size, in bytes, of the data chunks. This parameter specifies that a memory buffer, of length number of samples (NumSamples), collect data in multiples of WriteSize. By default, this parameter is 512 bytes, which is the typical disk sector size. Using a block size that is the same as the disk sector size provides optimal performance.</p> <p>If you experience a system crash, you can expect to lose an amount of data the size of WriteSize.</p> <p>For host or target scopes, this parameter has no effect.</p>	Yes
YLimit	Minimum and maximum y-axis values. This property can be set to 'auto'.	Yes

# xpctarget.xpcsc.set (scope object)

Property	Description	Writable
	For host or file scopes, this parameter has no effect.	

## Examples

Get a list of writable properties for a scope object.

```
sc1 = getscope(tg,1)
set(sc1)
ans=
    NumSamples: {}
    Decimation: {}
    TriggerMode: {5x1 cell}
    TriggerSignal: {}
    TriggerLevel: {}
    TriggerSlope: {4x1 cell}
    TriggerScope: {}
    TriggerSample: {}
    Signals: {}
    NumPrePostSamples: {}
    Mode: {5x1 cell}
    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

```
get | set | xpctarget.xpcsc.get (scope object) |
xpctarget.xpc.set (target application object) |
xpctarget.xpc.get (target application object)
```

# xpctarget.xpcsc.start (scope object)

---

**Purpose** Start execution of scope on target computer

**Syntax** MATLAB command line

```
start(scope_object_vector)
scope_object_vector.start
+scope_object_vector
start(getscope((target_object, signal_index_vector))
```

**Target computer 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 vector form [scope_object1, scope_object2], or the target object method getscope, which returns a scope_object vector.
signal_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 object. Starts a scope on the target computer represented by a scope object on the host computer. This method might not start data acquisition, which depends on the trigger settings. Before using this method, you must create a scope. 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

xpctarget.xpc.getscope | xpctarget.xpc.stop (target application object) | xpctarget.xpcsc.stop (scope object)

# xpctarget.xpcsc.stop (scope object)

---

**Purpose** Stop execution of scope on target computer

**Syntax** MATLAB command line

```
stop(scope_object_vector)
scope_object.stop
-scope_object
stop(getscope(target_object, signal_index_vector))
```

**Target computer 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.
signal_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. 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

## xpctarget.xpcsc.stop (scope object)

---

```
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**

```
xpctarget.xpc.getscope | xpctarget.xpc.stop (target  
application object) | xpctarget.xpc.start (target application  
object) | xpctarget.xpcsc.start (scope object)
```

# xpctarget.xpcsc.trigger

---

<b>Purpose</b>	Software-trigger start of data acquisition for scope(s)
<b>Syntax</b>	<b>MATLAB command line</b>  trigger(scope_object_vector) or scope_object_vector.trigger
<b>Arguments</b>	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.
<b>Description</b>	Method for a scope object. If the scope object property TriggerMode has a value of 'software', this function triggers the scope represented by the scope object to acquire the number of data points in the scope object property NumSamples.  Note that only scopes with type host store data in the properties scope_object.Time and scope_object.Data.
<b>Examples</b>	Set a single scope to software trigger, trigger the acquisition of one set of samples, and 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)
```

# xpctarget.xpcschoost Class

**Purpose** Control and access properties of host scopes

**Description** The scope gets a data package from the kernel, waits for an upload command from the host computer, and uploads the data to the host. The host computer displays the data using a scope viewer or other MATLAB functions.

## Methods

These methods are inherited from `xpctarget.xpcsc Class`.

Method	Description
<code>xpctarget.xpcsc.addsignals</code>	Add signals to scope represented by scope object
<code>xpctarget.xpcsc.get(scope object)</code>	Return property values for scope objects
<code>xpctarget.xpcsc.removesignals</code>	Remove signals from scope represented by scope object
<code>xpctarget.xpcsc.set(scope object)</code>	Change property values for scope objects
<code>xpctarget.xpcsc.start(scope object)</code>	Start execution of scope on target computer
<code>xpctarget.xpcsc.stop(scope object)</code>	Stop execution of scope on target computer
<code>xpctarget.xpcsc.trigger</code>	Software-trigger start of data acquisition for scope(s)

## Properties

These properties are inherited from `xpctarget.xpcsc Class`.

Property	Description	Writable
Application	Name of the Simulink model associated with this scope object.	No
Decimation	A number n, where every nth sample is acquired in a scope window.	Yes

Property	Description	Writable
NumPrePostSamples	For host or target scopes, this parameter is the 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 <b>TriggerMode</b> to 'FreeRun', this property has no effect on data acquisition.	Yes
NumSamples	<p>Number of contiguous samples captured during the acquisition of a data package. If the scope stops before capturing this number of samples, the scope has the collected data up to the end of data collection, then has zeroes for the remaining uncollected data. Note that you should know what type of data you are collecting, it is possible that your data contains zeroes.</p> <p>For file scopes, this parameter works in conjunction with the <b>AutoRestart</b> check box. If the <b>AutoRestart</b> box is selected, the file scope collects data up to <b>Number of Samples</b>, then starts over again, overwriting the buffer. If the <b>AutoRestart</b> box is not selected, the file scope collects data only up to <b>Number of Samples</b>, then stops.</p>	Yes
ScopeId	A numeric index, unique for each scope.	No
Signals	List of signal indices from the target object to display on the scope.	Yes
Status	Indicate 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'.	No

## xpctarget.xpcschoost Class

---

Property	Description	Writable
TriggerLevel	If TriggerMode is '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
TriggerMode	Trigger mode for a scope. Valid values are 'FreeRun' (default), 'Software', 'Signal', and 'Scope'.	Yes
TriggerSample	<p>If TriggerMode is 'Scope', then TriggerSample specifies which sample of the triggering scope the current scope should trigger on. For example, if TriggerSample is 0 (default), the current scope triggers on sample 0 (first sample acquired) of the triggering scope. This means that the two scopes will be perfectly synchronized. If TriggerSample is 1, the first sample (sample 0) of the current scope will be at the same instant as sample number 1 (second sample in the acquisition cycle) of the triggering scope.</p> <p>As a special case, setting TriggerSample to -1 means that the current scope is triggered at the end of the acquisition cycle of the triggering scope. Thus, the first sample of the triggering scope is acquired one sample after the last sample of the triggering scope.</p>	Yes
TriggerScope	If TriggerMode is 'Scope', identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered. You do this by setting the slave scope property TriggerScope to the scope index of the master scope.	Yes



## xpctarget.xpcschoost Class

Property	Description	Writable
TriggerSignal	If TriggerMode is 'Signal', identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property Signal.	Yes
TriggerSlope	If TriggerMode is 'Signal', indicates whether the trigger is on a rising or falling signal. Values are 'Either' (default), 'Rising', and 'Falling'.	Yes
Type	Determines whether the scope is displayed on the host computer or on the target computer. Values are 'Host', 'Target', and 'File'.	Yes

These properties are specific to class xpcschoost.

Property	Description	Writeable
Data	Contains the output data for a single data package from a scope. For target or file scopes, this parameter has no effect.	No
Time	Contains the time data for a single data package from a scope. For target or file scopes, this parameter has no effect.	No

# xpctarget.xpcsc Class

**Purpose** Control and access properties of target scopes

**Description** The kernel acquires a data package and the scope displays the data on the target computer screen. Depending on the setting of `DisplayMode`, the data may be displayed numerically or graphically by a redrawing, sliding, and rolling display.

## Methods

These methods are inherited from `xpctarget.xpcsc Class`.

Method	Description
<code>xpctarget.xpcsc.addsignals</code>	Add signals to scope represented by scope object
<code>xpctarget.xpcsc.get(scope object)</code>	Return property values for scope objects
<code>xpctarget.xpcsc.removesignals</code>	Remove signals from scope represented by scope object
<code>xpctarget.xpcsc.set(scope object)</code>	Change property values for scope objects
<code>xpctarget.xpcsc.start(scope object)</code>	Start execution of scope on target computer
<code>xpctarget.xpcsc.stop(scope object)</code>	Stop execution of scope on target computer
<code>xpctarget.xpcsc.trigger</code>	Software-trigger start of data acquisition for scope(s)

## Properties

These properties are inherited from `xpctarget.xpcsc Class`.

Property	Description	Writable
Application	Name of the Simulink model associated with this scope object.	No
Decimation	A number $n$ , where every $n$ th sample is acquired in a scope window.	Yes

Property	Description	Writable
NumPrePostSamples	For host or target scopes, this parameter is the 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 <b>TriggerMode</b> to 'FreeRun', this property has no effect on data acquisition.	Yes
NumSamples	<p>Number of contiguous samples captured during the acquisition of a data package. If the scope stops before capturing this number of samples, the scope has the collected data up to the end of data collection, then has zeroes for the remaining uncollected data. Note that you should know what type of data you are collecting, it is possible that your data contains zeroes.</p> <p>For file scopes, this parameter works in conjunction with the <b>AutoRestart</b> check box. If the <b>AutoRestart</b> box is selected, the file scope collects data up to <b>Number of Samples</b>, then starts over again, overwriting the buffer. If the <b>AutoRestart</b> box is not selected, the file scope collects data only up to <b>Number of Samples</b>, then stops.</p>	Yes
ScopeId	A numeric index, unique for each scope.	No
Signals	List of signal indices from the target object to display on the scope.	Yes
Status	Indicate 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'.	No

## xpctarget.xpcstg Class

---

Property	Description	Writable
TriggerLevel	If TriggerMode is '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
TriggerMode	Trigger mode for a scope. Valid values are 'FreeRun' (default), 'Software', 'Signal', and 'Scope'.	Yes
TriggerSample	<p>If TriggerMode is 'Scope', then TriggerSample specifies which sample of the triggering scope the current scope should trigger on. For example, if TriggerSample is 0 (default), the current scope triggers on sample 0 (first sample acquired) of the triggering scope. This means that the two scopes will be perfectly synchronized. If TriggerSample is 1, the first sample (sample 0) of the current scope will be at the same instant as sample number 1 (second sample in the acquisition cycle) of the triggering scope.</p> <p>As a special case, setting TriggerSample to -1 means that the current scope is triggered at the end of the acquisition cycle of the triggering scope. Thus, the first sample of the triggering scope is acquired one sample after the last sample of the triggering scope.</p>	Yes
TriggerScope	If TriggerMode is 'Scope', identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered. You do this by setting the slave scope property TriggerScope to the scope index of the master scope.	Yes

## xpctarget.xpcsctg Class

Property	Description	Writable
TriggerSignal	If TriggerMode is 'Signal', identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property Signal.	Yes
TriggerSlope	If TriggerMode is 'Signal', indicates whether the trigger is on a rising or falling signal. Values are 'Either' (default), 'Rising', and 'Falling'.	Yes
Type	Determines whether the scope is displayed on the host computer or on the target computer. Values are 'Host', 'Target', and 'File'.	Yes

These properties are specific to class xpcsctg.

Property	Description	Writeable
DisplayMode	For target scopes, indicate how a scope displays the signals. Values are 'Numerical', 'Redraw' (default), 'Sliding', and 'Rolling'.  For host or file scopes, this parameter has no effect.	Yes
Grid	Values are 'on' and 'off'.  For host or file scopes, this parameter has no effect.	Yes

## xpctarget.xpcsctg Class

Property	Description	Writeable
Mode	<p><b>Note</b> The Mode property will be removed in a future release.</p> <ul style="list-style-type: none"><li>• For target scopes, use DisplayMode.</li><li>• For file scopes, use WriteMode.</li><li>• For host scopes, this parameter has no effect.</li></ul>	Yes
YLimit	<p>Minimum and maximum y-axis values. This property can be set to 'auto'.</p> <p>For host or file scopes, this parameter has no effect.</p>	Yes

**Purpose** Test communication between host and target computers

**Syntax** MATLAB command line

```
xpctargetping
xpctargetping('mode', 'arg1', 'arg2')
```

**Arguments**

mode Optionally, enter the communication mode:

---

**Note** RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.

---

TCP/IP Enable TCP/IP connection with target computer.

RS232 Enable RS-232 connection with target computer.

arg1 Optionally, enter an argument based on the mode value:

IP If mode is 'TCP/IP', enter the IP address of the target computer.

COM If mode is 'RS232', enter the host COM port.

arg2 Optionally, enter an argument based on the mode value:

Port If mode is 'TCP/IP', enter the port number for the target computer.

Baud rate If mode is 'RS232', enter the baud rate for the connection between the host and target computer.

# xpctargetping

---

## Description

Pings the target computer from the host computer and returns either `success` or `failed`. If you have one target computer, or if you designate a target computer as the default one in your system, use the syntax

```
xpctargetping
```

If you have multiple target computers in your system, use the following syntax to identify the target computer to ping.

```
xpctargetping('mode', 'arg1', 'arg2')
```

This function returns `success` if the xPC Target kernel is loaded and running and communication is working between host and target.

This function works with both RS-232 and TCP/IP communication.

```
ans =  
success
```

## Examples

Check for communication between the host computer and target computer.

```
xpctargetping
```

If you have a serial connection with the target computer you want to check, use the following syntax.

```
xpctargetping('RS232', 'COM1', '115200')
```

## How To

- “Running the Confidence Test”



**Purpose** Open Real-Time xPC Target Spy window on host computer

**Syntax** **MATLAB command line**

```
xpctargetspy  
xpctargetspy(target_object)  
xpctargetspy('target_object_name')
```

**Arguments**

<code>target_object</code>	Variable name to reference the target object.
<code>target_object_name</code>	Target object name as specified in the xPC Target Explorer.

**Description** This graphical user interface (GUI) allows you to upload displayed data from the target computer. By default, `xpctargetspy` opens a Real-Time xPC Target Spy window for the target object, `tg`. If you have multiple target computers in your system, you can call the `xpctargetspy` function for a particular target object, `target_object`.

If you have one target computer, or if you designate a target computer as the default one in your system, use the syntax

```
xpctargetspy
```

If you have multiple target computers in your system, use `xpctarget.xpc` to create the additional target object first.

```
target_object=xpctarget.xpc('mode', 'arg1', 'arg2')
```

Then, use the following syntax.

```
xpctargetspy(target_object)
```

If you have a target computer object in the xPC Target Explorer, you can use the following syntax.

```
target_object=xpctarget.xpc('target_object_name')
```

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. You must explicitly request an update. To manually update the host screen with another target screen, move the pointer into the Real-Time xPC Target Spy window and right-click to select **Update xPC Target Spy**.
- If `TargetScope` is disabled, text output is transferred once every second to the host and displayed in the window.

### Examples

To open the Real-Time xPC Target Spy window for a default target computer, `tg`, in the MATLAB window, type

```
xpctargetspy
```

To open the Real-Time xPC Target Spy window for a target computer, `tg1`, in the MATLAB window, type

```
xpctargetspy(tg1)
```

If you have multiple target computers in your system, use `xpctarget.xpc` to create the additional target object, `tg2`, first.

```
tg2=xpctarget.xpc('RS232', 'COM1', '115200')
```

Then, use the following syntax.

```
xpctargetspy(tg2)
```

**Purpose** Test xPC Target installation

**Syntax** MATLAB command line

```
xpctest
xpctest('target_name')
xpctest('-noreboot')
xpctest('noreboot')
xpctest('target_name', 'noreboot')
xpctest('target_name', '-noreboot')
```

**Arguments**

'target_name'	Name of target computer to test.
'noreboot'	Only one possible option. Skips the reboot test. Use this option if the target hardware does not support software rebooting. Value is 'noreboot' or '-noreboot'.

**Description** xpctest is a series of xPC Target tests to check the functioning of the following xPC Target tasks:

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

xpctest('noreboot') or xpctest('-noreboot') skips the reboot test on the default target computer. Use this option if target hardware does not support software rebooting.

# xpctest

---

`xpctest('target_name')` runs the tests on the target computer identified by 'target\_name'.

`xpctest('target_name', 'reboot')` or `xpctest('target_name', '-reboot')` runs the tests on the target computer identified by 'target\_name', but skips the reboot test.

## Examples

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

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

To run `xpctest` on a specified target computer, for example TargetPC1, type

```
xpctest('TargetPC1')
```

## How To

- “Running the Confidence Test”
- “Test 1, Ping target PC 'TargetPC1' using system ping: FAILED” on page 13-2

**Purpose** Disconnect target computer from current client application

**Syntax** **MATLAB command line**

```
xpcwwenable  
xpcwwenable('target_obj_name')
```

**Description** Use this function to disconnect the target application from the MATLAB interface before you connect to the Web browser. You can also use this function to connect to the MATLAB interface after using a Web browser, or to switch to another Web browser.

`xpcwwenable('target_obj_name')` disconnects the target application on `target_obj_name`, for example 'TargetPC1', from the MATLAB interface.



# Configuration Parameters

---

This chapter deals with configuration parameters in xPC Target Explorer and in the MATLAB API.

## Setting Configuration Parameters

**In this section...**

“xPC Target options Pane” on page 29-3

“Automatically download application after building” on page 29-6

“Download to default target PC” on page 29-7

“Specify target PC name” on page 29-8

“Name of xPC Target object created by build process” on page 29-9

“Use default communication timeout” on page 29-10

“Specify the communication timeout in seconds” on page 29-11

“Execution mode” on page 29-12

“Real-time interrupt source” on page 29-13

“I/O board generating the interrupt” on page 29-14

“PCI slot (-1: autosearch) or ISA base address” on page 29-18

“Log Task Execution Time” on page 29-19

“Signal logging data buffer size in doubles” on page 29-20

“Enable profiling” on page 29-22

“Number of events (each uses 20 bytes)” on page 29-23

“Double buffer parameter changes” on page 29-24

“Load a parameter set from a file on the designated target file system” on page 29-26

“File name” on page 29-27

“Build COM objects from tagged signals/parameters” on page 29-28

“Generate CANape extensions” on page 29-29

“Include model hierarchy on the target application” on page 29-30

“Enable Stateflow animation” on page 29-31



## **xPC Target options Pane**

Set up general information about building target applications, including target, execution, data logging, and other options.

Select:

- ... Solver
- ... Data Import/Export
- + Optimization
- + Diagnostics
- ... Hardware Implementat...
- ... Model Referencing
- + Simulation Target
- Code Generation
  - ... Report
  - ... Comments
  - ... Symbols
  - ... Custom Code
  - ... Debug
  - ... xPC Target options
- + HDL Code Generation

**Target options**

Automatically download application after building

Download to default target PC

Name of xPC Target object created by build process

Use default communication timeout

**Execution options**

Execution mode

Real-time interrupt source

I/O board generating the interrupt

PCI slot (-1: autosearch) or ISA base address

**Data logging options**

Log Task Execution Time

Signal logging data buffer size in doubles

Enable profiling

Number of events (each uses 20 bytes)

**Application tunable parameter options**

Double buffer parameter changes

Load a parameter set from a file on the designated target file system

**Miscellaneous options**

Build COM objects from tagged signals/parameters

Generate CANape extensions

Include model hierarchy on the target application

Enable Stateflow animation

?
OK
Cancel
Help
Apply

## Configuration

To enable the xPC Target options pane, you must:

- 1 Select `xpctarget.tlc` or `xpctargetert.tlc` for the **System target file** parameter on the code generation pane.
- 2 Select **C** for the **Language** parameter on the code generation pane.

## Tips

- The default xPC Target options work for the generation of most target applications. If you want to customize the build of your target application, set the option parameters to suit your specifications.
- To access these parameters from the MATLAB command line, use:
  - `gcs` — To access the current model.
  - `set_param` — To set the parameter value.
  - `get_param` — To get the current value of the parameter.

## See Also

“xPC Target Options Configuration Parameter” on page 4-2

## Automatically download application after building

Enable Simulink Coder to build and download the target application to the target computer.

### Settings

**Default:** on



On

Builds and downloads the target application to the target computer.



Off

Builds the target application, but does not download it to the target computer.

### Command-Line Information

**Parameter:** xPCisDownloadable

**Type:** string

**Value:** 'on' | 'off'

**Default:** 'on'

### See Also

“Building and Downloading Target Application”

## Download to default target PC

Direct Simulink Coder to download the target application to the default target computer.

### Settings

**Default:** on



On

Downloads the target application to the default target computer. Your default target computer must be set up in xPC Target Explorer.



Off

Enables the **Specify target PC name** field so that you can enter the target computer to which to download the target application.

### Tip

This assumes that you configured a default target computer through the xPC Target Explorer.

### Dependency

This parameter enables **Specify target PC name**.

### Command-Line Information

**Parameter:** xPCisDefaultEnv

**Type:** string

**Value:** 'on' | 'off'

**Default:** 'on'

### See Also

- “Network Communication”
- “Serial Communication”

### Specify target PC name

Specify a target computer name for your target application.

### Settings

''

### Tip

The target computer name appears in the xPC Target Explorer as the target computer node, for example TargetPC1.

### Dependencies

This parameter is enabled by **Download to default target PC**.

### Command-Line Information

**Parameter:** xPCTargetPCEnvName

**Type:** string

**Value:** Any valid target computer

**Default:** ''

### See Also

“xPC Target Explorer” on page 4-3

## **Name of xPC Target object created by build process**

Enter the name of the target object created by the build process.

### **Settings**

**Default:** tg

### **Tip**

Use this name when you work with the target object through the command-line interface.

### **Command-Line Information**

**Parameter:** RL320bjectName

**Type:** string

**Value:** 'tg' | valid target object name

**Default:** 'tg'

### **See Also**

“Target Driver Objects” on page 7-2

### Use default communication timeout

Direct xPC Target software to wait 5 (default) seconds for the target application to be downloaded to the target computer.

#### Settings

**Default:** on



On

Waits the default amount of seconds (5) for the target application to be downloaded to the target computer.



Off

Enables the **Specify the communication timeout in seconds** field so that you can enter the maximum length of time in seconds you want to wait for a target application to be downloaded to the target computer.

#### Dependencies

This parameter enables **Specify the communication timeout in seconds**.

#### Command-Line Information

**Parameter:** xPCisModelTimeout

**Type:** string

**Value:** 'on' | 'off'

**Default:** 'on'

#### See Also

“How Do I Increase the Download Timeout Value?” on page 20-4



## Specify the communication timeout in seconds

Specify a timeout, in seconds, to wait for the target application to download to the target computer.

### Settings

**Default:** 5

### Tip

Enter the maximum length of time in seconds you want to allow the xPC Target software to wait for the target application to download to the target computer. If the target application is not downloaded within this time frame, the software generates an error.

### Dependencies

This parameter is enabled by **Use default communication timeout**.

### Command-Line Information

**Parameter:** xPCModelTimeoutSecs

**Type:** string

**Value:** Any valid number of seconds

**Default:** '5'

### See Also

“How Do I Increase the Download Timeout Value?” on page 20-4

### Execution mode

Specify target application execution mode.

#### Settings

**Default:** Real-Time

Real-Time

Executes target application in real time.

Freerun

Runs the target application as fast as possible.

#### Command-Line Information

**Parameter:** RL32ModeModifier

**Type:** string

**Value:** 'Real-Time' | 'Freerun'

**Default:** 'Real-Time'

#### See Also

“Entering Simulation Parameters”

## Real-time interrupt source

Select a real-time interrupt source from the I/O board.

### Settings

**Default:** Timer

Timer

Specifies that the board interrupt source is a timer.

Auto (PCI only)

Enables the xPC Target software to automatically determine the IRQ that the BIOS assigned to the board and use it.

3 to 15

Specifies that the board interrupt source is an IRQ number on the board.

### Tips

- The Auto (PCI only) option is available only for PCI boards. If you have an ISA board (PC 104 or onboard parallel port), you must set the IRQ manually.
- The xPC Target software treats PCI parallel port plug-in boards like ISA boards. For PCI parallel port plug-in boards, you must set the IRQ manually.
- Multiple boards can share the same interrupt number.

### Command-Line Information

**Parameter:** RL32IRQSourceModifier

**Type:** string

**Value:** 'Timer' | Auto (PCI only) | '3' | '4' | '5' | '6' | '7' | '8' | '9' | '10' | '11' | '12' | '13' | '14' | '15'

**Default:** 'Timer'

### See Also

“Entering Simulation Parameters”

## **I/O board generating the interrupt**

Specify the board interrupt source.

### **Settings**

**Default:** None/Other

ATI-RP-R5

Specifies that the interrupt source is an ATI-RP-R5 board.

AudioPMC+

Specifies that the interrupt source is the Bittware AudioPMC+ audio board.

Bitflow NEON

Specifies that the interrupt source is the Bitflow NEON video board.

CB\_CIO-CTR05

Specifies that the interrupt source is the Measurement Computing CIO-CTR05 board.

CB\_PCI-CTR05

Specifies that the interrupt source is the Measurement Computing PCI-CTR05 board.

Diamond\_MM-32

Specifies that the interrupt source is the Diamond Systems MM-32 board.

FastComm 422/2-PCI

Specifies that the interrupt source is the FastComm 422/2-PCI board.

FastComm 422/2-PCI-335

Specifies that the interrupt source is the FastComm 422/2-PCI-335 board.

FastComm 422/4-PCI-335

Specifies that the interrupt source is the FastComm 422/4-PCI-335 board.

GE\_Fanuc(VMIC)\_PCI-5565

Specifies that the interrupt source is the GE Fanuc VMIC PCI-5565 board.

**General Standards 24DSI12**

Specifies that the interrupt source is the General Standards 24DSI12 board.

**Parallel\_Port**

Specifies that the interrupt source is the parallel port of the target computer.

**Quatech DSCP-200/300**

Specifies that the interrupt source is the Quatech DSCP-200/300 board.

**Quatech ESC-100**

Specifies that the interrupt source is the Quatech ESC-100 board.

**Quatech QSC-100**

Specifies that the interrupt source is the Quatech QSC-100 board.

**Quatech QSC-200/300**

Specifies that the interrupt source is the Quatech QSC-200/300 board.

**RTD\_DM6804**

Specifies that the interrupt source is the Real-Time Devices DM6804 board.

**SBS\_25x0\_ID\_0x100**

Specifies that the interrupt source is an SBS Technologies shared memory board associated with ID 0x100.

**SBS\_25x0\_ID\_0x101**

Specifies that the interrupt source is an SBS Technologies shared memory board associated with ID 0x101.

**SBS\_25x0\_ID\_0x102**

Specifies that the interrupt source is an SBS Technologies shared memory board associated with ID 0x102.

**SBS\_25x0\_ID\_0x103**

Specifies that the interrupt source is an SBS Technologies shared memory board associated with ID 0x103.

**Scramnet\_SC150+**

Specifies that the interrupt source is the Systran Scramnet+ SC150 board.

**Softing\_CAN-AC2-104**

Specifies that the interrupt source is the Softing CAN-AC2-104 board.

Softing\_CAN-AC2-PCI

Specifies that the interrupt source is the Softing CAN-AC2-PCI board.

Speedgoat\_IO301

Specifies that the interrupt source is the Speedgoat IO301 FPGA board.

Speedgoat\_IO302

Specifies that the interrupt source is the Speedgoat IO302 FPGA board.

Speedgoat\_IO303

Specifies that the interrupt source is the Speedgoat IO303 FPGA board.

Speedgoat\_IO311

Specifies that the interrupt source is the Speedgoat IO311 FPGA board.

Speedgoat\_IO312

Specifies that the interrupt source is the Speedgoat IO312 FPGA board.

Speedgoat\_IO313

Specifies that the interrupt source is the Speedgoat IO313 FPGA board.

Speedgoat\_IO314

Specifies that the interrupt source is the Speedgoat IO314 FPGA board.

Speedgoat\_IO325

Specifies that the interrupt source is the Speedgoat IO325 FPGA board.

UEI\_MF<sub>x</sub>

Specifies that the interrupt source is a United Electronic Industries UEI-MF series board.

None/Other

Specifies that the I/O board has no interrupt source.

## Command-Line Information

**Parameter:** xPCIRQSourceBoard

**Type:** string

**Value:** 'ATI-RP-R5' |  
'AudioPMC+' |  
'Bitflow NEON' |  
'CB\_CIO-CTR05' |  
'CB\_PCI-CTR05' |  
'Diamond\_MM-32' |  
'FastComm 422/2-PCI' |

```
'FastComm 422/2-PCI-335' |  
'FastComm 422/4-PCI-335' |  
'GE_Fanuc(VMIC)_PCI-5565' |  
'General Standards 24DSI12' |  
'Parallel_Port' |  
'Quatech DSCP-200/300' |  
'Quatech ESC-100' |  
'Quatech QSC-100' |  
'Quatech QSC-200/300' |  
'RTD_DM6804' |  
'SBS_25x0_ID_0x100' |  
'SBS_25x0_ID_0x101' |  
'SBS_25x0_ID_0x102' |  
'SBS_25x0_ID_0x103' |  
'Scramnet_SC150+' |  
'Softing_CAN-AC2-104' |  
'Softing_CAN-AC2-PCI' |  
'Speedgoat_I0301' |  
'Speedgoat_I0302' |  
'Speedgoat_I0303' |  
'Speedgoat_I0311' |  
'Speedgoat_I0312' |  
'Speedgoat_I0313' |  
'Speedgoat_I0314' |  
'Speedgoat_I0325' |  
'UEI_MFx' |  
'None/Other'  
Default: 'None/Other'
```

## See Also

“Entering Simulation Parameters”

## **PCI slot (-1: autosearch) or ISA base address**

Enter the slot number or base address for the I/O board generating the interrupt.

### **Settings**

**Default:** -1

The PCI slot can be either -1 (let the xPC Target software determine the slot number) or of the form [bus, slot].

The base address is a hexadecimal number of the form 0x300.

### **Tip**

To determine the bus and PCI slot number of the boards in the target computer:

- In the xPC Target Explorer, connect to the target computer in question and expand the `PCI Devices` node.
- In the MATLAB Window, type `getxpcpci`.

### **Command-Line Information**

**Parameter:** `xPCIOIRQSlot`

**Type:** string

**Value:** '-1' | hexadecimal value

**Default:** '-1'

### **See Also**

“xPC Target Options Configuration Parameter” on page 4-2

“PCI Bus I/O Devices”



## Log Task Execution Time

Log task execution times to the target object property `tg.TETlog`.

### Settings

**Default:** on



On

Logs task execution times to the target object property `tg.TETlog`.



Off

Does not log task execution times to the target object property `tg.TETlog`.

### Command-Line Information

**Parameter:** RL32LogTETModifier

**Type:** string

**Value:** 'on' | 'off'

**Default:** 'on'

### See Also

“xPC Target Options Configuration Parameter” on page 4-2

“Signal Logging” on page 5-57

## Signal logging data buffer size in doubles

Enter the maximum number of sample points to save before wrapping.

### Settings

**Default:** 100000

The maximum value for this option can be no more than the available target computer memory, which the xPC Target software also uses to hold other items.

### Tips

- Target applications use this buffer to store the time, states, outputs, and task execution time logs as defined in the Simulink model.
- The maximum value for this option derives from available target computer memory, which the xPC Target software also uses to hold other items. For example, in addition to signal logging data, the software also uses the target computer memory for the xPC Target kernel, target application, and scopes.

For example, assume the model `my_xpc_osc2.mdl` has six data items (one time, two states, two outputs, and one task execution time (TET)). If you enter a buffer size of 100000, the target object property `tg.MaxLogSamples` is calculated as  $\text{floor}(100000) / 6 = 16666$ . After saving 16666 sample points, the buffer wraps and further samples overwrite the older ones.

- If you enter a logging buffer size larger than the available RAM on the target computer, after downloading and initializing the target application, the target computer displays a message, `ERROR: allocation of logging memory failed`. In this case you need to install more RAM or reduce the buffer size for logging. In any case you must reboot the target computer. To calculate the maximum buffer size you might have for your target application logs, divide the amount of available RAM on your target computer by 8. Enter that value for the **Signal logging data buffer size in doubles** value.

### Command-Line Information

**Parameter:** `RL32LogBufSizeModifier`

**Type:** string

**Value:** '100000' | any valid memory size

**Default:** '100000'

**See Also**

“xPC Target Options Configuration Parameter” on page 4-2

### Enable profiling

Enable profiling and visual presentation of target computer execution.

#### Settings

**Default:** off

On  
Profile target computer execution.

Off  
Do not profile target computer execution.

#### Tips

- Before building and downloading a model, select this check box to observe the target computer thread execution.
- If you are using multiple CPU cores on a target computer, select **Enabling profiling** to verify that the xPC Target software is actually executing on the multiple CPU cores.

---

**Tip** For more on configuring your model for concurrent execution, see “Configuring Models for Targets with Multicore Processors”.

---

#### Command-Line Information

**Parameter:** xPCTaskExecutionProfile

**Type:** string

**Value:** 'on' | 'off'

**Default:** 'off'

#### See Also

Chapter 26, “Tuning Performance”

## Number of events (each uses 20 bytes)

Enter the maximum of events to log for the profiling tool.

### Settings

**Default:** 5000

The maximum number of events to be logged for the profiling tool.

### Tips

- An event is the start or end of an interrupt or iteration of the model. For example, one sample can have four events: the beginning and end of an interrupt, and the beginning and end of an iteration.
- Use this parameter in conjunction with the **Enable profiling** parameter.
- Each event contains information such as the CPU ID, model thread ID (TID), event ID, and time stamp readings. Each event occupies 20 bytes.

### Command-Line Information

**Parameter:** xPCRL32EventNumber

**Type:** string

**Value:** any valid number of events

**Default:** '5000'

### See Also

Chapter 26, “Tuning Performance”

## Double buffer parameter changes

Use a double buffer for parameter tuning. This enables parameter tuning so that the process of changing parameters in the target application uses a double buffer.

### Settings

**Default:** off

On  
Changes parameter tuning to use a double buffer.

Off  
Suppresses double buffering of parameter changes in the target application.

### Tips

- When a parameter change request is received, the new value is compared to the old one. If the new value is identical to the old one, it is discarded, and if different, queued.
- At the start of execution of the next sample of the real-time task, all queued parameters are updated. This means that parameter tuning affects the task execution time (TET), and the very act of parameter tuning can cause a CPU overload error.
- Double buffering leads to a more robust parameter tuning interface, but it increases Task Execution Time (TET) and the higher probability of overloads. Under typical conditions, keep double buffering off (default).

### Command-Line Information

**Parameter:** xpcDb1Buff

**Type:** string

**Value:** 'on' | 'off'

**Default:** 'off'

**See Also**

“xPC Target Options Configuration Parameter” on page 4-2

## Load a parameter set from a file on the designated target file system

Automatically load a parameter set from a file on the designated target computer file system.

### Settings

**Default:** off



On

Enable the automatic loading of a parameter set from the file specified by **File name** on the designated target computer file system.



Off

Suppress the automatic loading of a parameter set from a file on the designated target computer file system.

### Dependencies

This parameter enables **File name**.

### Command-Line Information

**Parameter:** xPCLoadParamSetFile

**Type:** string

**Value:** 'on' | 'off'

**Default:** 'off'

### See Also

“xPC Target Options Configuration Parameter” on page 4-2

“Saving and Reloading Application Parameters with the MATLAB Interface” on page 5-76



## File name

Specify the target computer file name from which to load the parameter set.

## Settings

''

## Tip

If the named file does not exist, the software loads the parameter set built with the model.

## Dependencies

This parameter is enabled by **Load a parameter set from a file on the designated target file system**.

## Command-Line Information

**Parameter:** xPCOnTgtParamSetFileName

**Type:** string

**Value:** Any valid file name

**Default:** ''

## See Also

“xPC Target Options Configuration Parameter” on page 4-2

## **Build COM objects from tagged signals/parameters**

Enable build process to create a model-specific COM library file.

### **Settings**

**Default:** off



On

Creates a model-specific COM library file, <model\_name>COMiface.dll.



Off

Does not create a model-specific COM library file.

### **Tip**

Use the model-specific COM library file to create custom GUIs with Visual Basic® or other tools that can use COM objects.

### **Command-Line Information**

**Parameter:** xpcObjCom

**Type:** string

**Value:** 'on' | 'off'

**Default:** 'off'

### **See Also**

“Creating the Target Application and Model-Specific COM Library”

## Generate CANape extensions

Enable target applications to generate data, such as that for A2L, for Vector CANape.

### Settings

**Default:** off



On

Enables target applications to generate data, such as that for A2L, for Vector CANape.



Off

Does not enable target applications to generate data, such as that for A2L, for Vector CANape.

### Command-Line Information

**Parameter:** xPCGenerateASAP2

**Type:** string

**Value:** 'on' | 'off'

**Default:** 'off'

### See Also

“Configuring the Vector CANape Device” on page 2-7

## **Include model hierarchy on the target application**

Includes the Simulink model hierarchy as part of the target application.

### **Settings**

**Default:** off



On

Includes the model hierarchy as part of the target application.



Off

Excludes the model hierarchy from the target application.

### **Tips**

Including the model hierarchy in the target application:

- Lets you connect to the target computer from xPC Target Explorer without being in the target application build directory.
- Can increase the size of the target application, depending on the size of the model.

### **Command-Line Information**

**Parameter:** xPCGenerateXML

**Type:** string

**Value:** 'on' | 'off'

**Default:** 'off'

### **See Also**

“Signal Monitoring with xPC Target Explorer” on page 5-2

## Enable Stateflow animation

Enables visualization of Stateflow chart animation.

### Settings

**Default:** off



On

Enables visualization of Stateflow chart animation.



Off

Disables visualization of Stateflow chart animation.

### Command-Line Information

**Parameter:** xPCEnableSFAnimation

**Type:** string

**Value:** 'on' | 'off'

**Default:** 'off'

### See Also

“Animating Stateflow Charts” on page 5-14



## A

- application parameters
  - saving and reloading 5-76

## B

- block parameters
  - parameter tuning with external mode 5-73
- booting
  - troubleshooting 17-4 to 17-5
- build process
  - troubleshooting 20-2

## C

- changing environment properties
  - CLI 4-19
  - xPC Target Explorer 4-16
- changing parameters
  - using target object properties 5-70
  - xPC Target commands 5-70
- code generation options
  - reference 4-2
- command-line interface
  - scope object 7-8
  - target objects 7-2
  - target PC 10-1
- configuration parameters
  - pane 29-3
    - Automatically download application after building 29-6
    - Build COM objects from tagged signals/parameters 29-28
    - Double buffer parameter changes 29-24
    - Download to default target PC 29-7
    - Enable Stateflow Animation 29-31
    - Execution mode 29-12
    - File name 29-27
    - Generate CANape extensions 29-29
    - I/O board generating the interrupt 29-14

- Include model hierarchy on the target application 29-30
- Load a parameter set from a file on the designated target file system 29-26
- Log Task Execution Time 29-19
- Name of xPC Target object created by build process 29-9
- PCI slot/ISA base address 29-18
- Real-time interrupt source 29-13
- Signal logging data buffer size in doubles 29-20
- Specify target PC name 29-8
- Specify the communication timeout in seconds 29-11
- Use default communication timeout 29-10

- CPU overloads
  - troubleshooting 24-4

## D

- data logging
  - with MATLAB 5-60
  - with Web browser 5-64
- default target computer
  - introduction 4-8

## E

- entering environment properties
  - xPC Target Explorer 4-16
- environment collection objects
  - target PC 4-10
- environment properties
  - changing through CLI 4-19
  - changing through xPC Target Explorer 4-16
  - list 4-14
  - updating through CLI 4-19
  - updating through xPC Target Explorer 4-16
- exporting and importing

- xPC Target Explorer 4-31

external mode

- parameter tuning 5-73

## F

file system objects

- methods 27-10
- `xpctarget.fs` introduction 8-4

file systems

- introduction 8-2
- target computer 8-2

Fortran

- S- function wrapper 3-8
- wrapper S-function 3-8
- xPC Target 3-2

FTP objects

- `xpctarget.ftp` introduction 8-4

functions 5-40

- changing parameters 5-70
- signal logging 5-60
- signal monitoring 5-9

## G

- getting list of environment properties 4-14
- getting parameter properties 5-70
- getting signal properties 5-9

## H

host scope viewer

- xPC Target Explorer 5-31

## I

inlined parameters

- tuning with MATLAB 5-84
- tuning with xPC Target Explorer 5-81

interrupt mode

- introduction 6-1

## L

list

- environment properties 4-14

## M

MathWorks

- technical support 25-5

MATLAB 5-40

- parameter tuning 5-70
- signal logging 5-60
- signal monitoring 5-9

methods

- file system object 27-10

monitoring signals

- referenced models 5-9
- xPC Target Explorer 5-2

monitoring Stateflow states

- MATLAB interface 5-10

## P

parameter tuning 5-73

- overview 5-66

- Web browser 5-75

- with MATLAB 5-70

- with Simulink external mode 5-73

parameters

- changing with commands 5-70

- inlining 5-79

- tuning with external mode 5-73

- tuning with MATLAB 5-70

- tuning with Web browser 5-75

polling mode

- introduction 6-1

- setting up 6-7

properties

- changing environment 4-19

- environment list 4-14

- updating environment 4-19



**R**

readxpcfile 8-14  
 referenced models  
   monitoring signals 5-9

**S**

S-Function  
   xPC Target 3-2  
 saving and reloading application parameters  
   with MATLAB 5-76  
 saving and reloading application sessions 5-38  
 scope objects  
   command-line interface 7-8  
   commands 7-8  
   list of properties with files 5-46  
   list of properties with targets 5-42  
   methods, *see* commands 7-8  
   properties 7-8  
 scopes  
   creating 5-17  
   software triggering 5-29  
   stopping 5-28  
 Setup window  
   using 4-14  
 signal logging  
   overview 5-57  
   with MATLAB 5-60  
   with Web browser 5-64  
   xPC Target Explorer 5-57  
 signal monitoring  
   with MATLAB 5-9  
 signal tracing  
   with MATLAB 5-40  
   with Simulink external mode 5-51  
   with Web browser 5-55  
   with xPC Target scope blocks 5-49  
 signals  
   adding 5-24  
 Simulink Coder

code generation options 4-2

Simulink external mode  
   parameter tuning 5-73  
   signal tracing 5-51  
 Stateflow states  
   monitoring 5-10

**T**

target application  
   saving and reloading sessions 5-38  
 target computer  
   copying files with `xpctarget.ftp` 8-8  
   disk information retrieval with  
     `xpctarget.fs` 8-18  
   file content retrieval with  
     `xpctarget.fs` 8-12  
   file conversion with `xpctarget.fs` 8-14  
   file information retrieval with  
     `xpctarget.fs` 8-17  
   file removal with `xpctarget.fs` 8-15  
   file retrieval with `xpctarget.ftp` 8-8  
   folder listings with `xpctarget.ftp` 8-7  
   list of open files with `xpctarget.fs` 8-16  
 target object properties  
   file scopes 5-45  
 target objects  
   changing parameters 5-70  
   command-line interface 7-2  
   commands 7-2  
   list of properties with files 5-45  
   methods, *see* commands 7-2  
   parameter properties 5-70  
   properties 7-2  
   signal properties 5-9  
 target PC  
   command-line interface 10-1  
   environment collection objects 4-10  
   manipulating scope object properties 10-6  
   manipulating scope objects 10-4

- manipulating target object properties 10-3
- using target application methods 10-2
- task execution time (TET)
  - average 28-127 28-141
  - definition 5-63
  - logging 28-132 28-146
  - maximum 28-129 28-142
  - minimum 28-129 28-142
  - with the `getlog` function 28-147
- TET. *See* task execution time
- timeout value
  - changing 20-4
- tracing signals
  - xPC Target Explorer 5-16
- troubleshooting
  - application execution 13-16
  - BIOS settings 15-2
  - boot disk 25-4
  - boot image 25-4
  - boot process 17-4 to 17-5
  - booting target 17-1
  - build 13-9
  - build process 20-2
  - changed stop time 21-6
  - communication 13-2 13-5 13-9 13-12 13-14
  - communication issues 16-2
  - compilation 13-9
  - confidence test 12-1 13-1
  - connection lost 16-4 16-6 16-8
  - CPU Overload 24-4
  - CPU overloads 24-4
  - custom device drivers 18-3
  - device drivers 18-3
  - different sample times 21-3
  - download 13-9 13-14 20-1
  - Error -10 23-2
  - execution 13-15 21-1
  - file system disabled 15-4
  - general I/O 15-8
  - `getxpcpci` 15-6
  - host configuration 14-1
  - host PC MATLAB halted 14-2
  - I/O driver errors 16-8
  - I/O drivers 18-1
  - installation, configuration, and tests 12-1
  - invalid file ID 23-2
  - lost connection 16-4 16-6 16-8
  - model compilation 19-1
  - multiple Ethernet cards 16-6
  - new releases 25-4
  - parameters 13-15 to 13-16 22-1
  - PCI board slot and bus 15-6
  - PCI boards 15-6
  - performance 24-1
  - sample time differences 21-3
  - sample times 21-3
  - signals 13-15 to 13-16 23-1
  - slow initialization time 16-4
  - stack size 15-5
  - standalone xPC Target application 19-2
  - stop time change 21-6
  - support 25-1
  - tagging virtual blocks 23-3
  - target application 13-15 to 13-16
  - target application build 19-1 20-1
  - target boot 13-7
  - target computer hardware 15-1
  - target configuration 13-2 13-5 13-7 13-9
    - 13-12 13-14 to 13-16
  - target PC halted 17-6
  - target PC monitor view 21-2
  - test failures 13-1
  - timeout value 20-4
  - updated xPC Target releases 25-3
  - virtual block tagging 23-3
  - xPC Target PC unable to boot 17-2
  - `xpctargetspy` 21-2
  - `xpctest` 12-1 13-1
- tuning parameters
  - xPC Target Explorer 5-66

**U**

- updating environment properties through  
CLI 4-19
- updating environment properties through xPC  
Target Explorer 4-16
- using setup window 4-14
- using xPC Target setup window 4-14

**W**

- Web browser 5-55
  - connecting 11-2
  - parameter tuning 5-75
  - signal logging 5-64

**X**

- xPC Target
  - application download 20-1
  - application execution 21-1
  - application parameters 22-1
  - application performance 24-1
  - application signals 23-1
  - host-target communication 16-1
  - MathWorks support 25-1
  - modeling 18-1
  - procedure 12-1
  - target boot disk 17-1
  - troubleshooting 12-1 13-1 14-1 15-1 16-1  
17-1 18-1 20-1 21-1 22-1 23-1 24-1 25-1
  - Web browser 11-1

- xPC Target Explorer
  - adding signals 5-24
  - configuring the host scope viewer 5-31
  - creating scopes 5-17
  - exporting and importing 4-31
  - introduction 4-3
  - logging 5-57
  - menu bar 4-6
  - monitoring signals 5-2
  - shortcut keys 4-6
  - stopping scopes 5-28
  - toolbar 4-6
  - tracing signals 5-16
  - tuning parameters 5-66
- xPC Target scope blocks 5-49
- xPC Target Setup window 4-14
- xpctarget.fs
  - creation 8-4
  - introduction 8-2
  - methods 27-10
  - overview 8-10
- xpctarget.fsbase
  - methods 27-10
- xpctarget.ftp
  - creation 8-4
  - introduction 8-2
  - methods 27-10
  - overview 8-5
- xpctcp2ser 11-5