

# xPC Target

## For Use with Real-Time Workshop<sup>®</sup>

- Modeling
- Simulation
- Implementation

## How to Contact The MathWorks:



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



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



508-647-7000      Phone



508-647-7001      Fax



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

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

### *xPC Target User's Guide*

© COPYRIGHT 1999 - 2004 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.

MATLAB, Simulink, Stateflow, Handle Graphics, and Real-Time Workshop are registered trademarks, and TargetBox is a trademark of The MathWorks, Inc.

Other product or brand names are trademarks or registered trademarks of their respective holders.

Printing 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+)

## Target and Scope Objects

1

<b>Target Objects</b> .....	1-2
What Is a Target Object? .....	1-2
<b>Scope Objects</b> .....	1-3
What Is a Scope Object? .....	1-3

## Targets and Scopes in the MATLAB Interface

2

<b>Working with Target Objects</b> .....	2-2
Creating Target Objects .....	2-2
Deleting Target Objects .....	2-3
Displaying Target Object Properties .....	2-3
Setting Target Object Properties from the Host PC .....	2-4
Resetting Target Application Parameters to Previous Values .	2-5
Getting the Value of a Target Object Property .....	2-6
Using the Method Syntax with Target Objects .....	2-7
<b>Working with Scope Objects</b> .....	2-8
Displaying Scope Object Properties for a Single Scope .....	2-8
Displaying Scope Object Properties for All Scopes .....	2-9
Setting the Value of a Scope Property .....	2-9
Getting the Value of a Scope Property .....	2-10
Using the Method Syntax with Scope Objects .....	2-11
Acquiring Signal Data with Scopes of Type File .....	2-12
Advanced Data Acquisition Topics .....	2-13

<b>Monitoring Signals</b> .....	<b>3-2</b>
Signal Monitoring with xPC Target Explorer .....	<b>3-2</b>
Signal Monitoring with MATLAB .....	<b>3-5</b>
<b>Signal Tracing</b> .....	<b>3-7</b>
Signal Tracing with xPC Target Explorer .....	<b>3-7</b>
Signal Tracing with MATLAB .....	<b>3-17</b>
Signal Tracing with xPC Target Scope Blocks .....	<b>3-25</b>
Signal Tracing with a Web Browser .....	<b>3-26</b>
<b>Signal Logging</b> .....	<b>3-28</b>
Signal Logging with xPC Target Explorer .....	<b>3-28</b>
Signal Logging with MATLAB .....	<b>3-30</b>
Signal Logging with a Web Browser .....	<b>3-34</b>
<b>Parameter Tuning</b> .....	<b>3-35</b>
Parameter Tuning with xPC Target Explorer .....	<b>3-35</b>
Parameter Tuning with MATLAB .....	<b>3-38</b>
Parameter Tuning with Simulink External Mode .....	<b>3-40</b>
Parameter Tuning with a Web Browser .....	<b>3-42</b>
Saving and Reloading Application Parameters with MATLAB	<b>3-43</b>

# 4

<b>Introduction</b> .....	4-2
Before You Start .....	4-2
<b>Specifying Target PC Configurations</b> .....	4-4
Configuring Environment Parameters for Target PCs .....	4-4
Creating a Target PC Boot Disk .....	4-5
<b>Preparing Target PCs for xPC Target System Use</b> .....	4-8
Installing Software, Booting a Target PC, and Testing the Target PC .....	4-8
<b>Downloading and Running Target Applications</b> .....	4-10
Changing the Prebuilt Target Application (DLM) Directory .	4-10
Downloading Target Applications to a Target PC .....	4-11
Running the Target Application .....	4-13
<b>Control with xPC Target Explorer</b> .....	4-15
Manipulating Target Application Properties .....	4-15
<b>Menu Bar and Toolbar Contents and Shortcut Keys</b> .....	4-17

## Embedded Option

# 5

<b>Introduction</b> .....	5-2
<b>xPC Target Embedded Option Modes</b> .....	5-3
DOSLoader Mode Overview .....	5-4
StandAlone Mode Overview .....	5-5
Restrictions .....	5-7
<b>Embedded Option Setup</b> .....	5-9
Updating the xPC Target Environment .....	5-9
Creating a DOS System Disk .....	5-11

<b>DOSLoader Target Setup</b> .....	<b>5-12</b>
Updating Environment Properties and Creating a Boot Disk	<b>5-12</b>
Copying the Kernel to Flash Memory .....	<b>5-14</b>
Creating a Target Application for DOSLoader Mode .....	<b>5-16</b>
<b>Stand-Alone Target Setup</b> .....	<b>5-17</b>
Updating Environment Properties .....	<b>5-17</b>
Adding Target Scope Blocks to Stand-Alone Applications ...	<b>5-18</b>
Creating a Kernel/Target Application .....	<b>5-21</b>
Copying the Kernel/Target Application to Flash Disk .....	<b>5-22</b>

## Software Environment and Demos

### 6

<b>Using Environment Properties and Functions</b> .....	<b>6-2</b>
Getting a List of Environment Properties .....	<b>6-2</b>
Changing Environment Properties with a Graphical Interface	<b>6-3</b>
Changing Environment Properties with a Command-Line Interface .....	<b>6-5</b>
<b>xPC Target Demos</b> .....	<b>6-6</b>
To Locate or Edit a Demo Script .....	<b>6-6</b>

## Using the Target PC Command-Line Interface

### 7

<b>Target PC Command-Line Interface</b> .....	<b>7-2</b>
Using Target Application Methods on the Target PC .....	<b>7-2</b>
Manipulating Target Object Properties from the Target PC ..	<b>7-3</b>
Manipulating Scope Objects from the Target PC .....	<b>7-4</b>
Manipulating Scope Object Properties from the Target PC ...	<b>7-6</b>
Aliasing with Variable Commands on the Target PC .....	<b>7-6</b>

## Working with Target PC Files and File Systems

# 8

<b>Introduction</b> .....	<b>8-2</b>
<b>FTP and File System Objects</b> .....	<b>8-4</b>
<b>Using <code>xpctarget.ftp</code> Objects</b> .....	<b>8-5</b>
Listing the Contents of the Target PC Directory .....	<b>8-5</b>
Retrieving a File from the Target PC to the Host PC .....	<b>8-6</b>
Copying a File from the Host PC to the Target PC .....	<b>8-7</b>
<b>Using <code>xpctarget.fs</code> Objects</b> .....	<b>8-8</b>
Retrieving the Contents of a File from the Target PC to the Host PC .....	<b>8-9</b>
Removing a File from the Target PC .....	<b>8-11</b>
Getting a List of Open Files on the Target PC .....	<b>8-11</b>
Getting Information about a File on the Target PC .....	<b>8-12</b>
Getting Information about a Disk on the Target PC .....	<b>8-13</b>

## Graphical User Interfaces

# 9

<b>xPC Target Interface Blocks to Simulink Models</b> .....	<b>9-2</b>
Simulink User Interface Model .....	<b>9-2</b>
Creating a Custom Graphical Interface .....	<b>9-4</b>
To xPC Target Block .....	<b>9-4</b>
From xPC Target Block .....	<b>9-6</b>

<b>Interface with Dials &amp; Gauges Blockset</b> .....	<b>9-8</b>
Introduction to the Dials & Gauges Blockset .....	<b>9-8</b>
Target Application Model Description .....	<b>9-11</b>
Creating a Target Application Model .....	<b>9-12</b>
Description of the User Interface Model .....	<b>9-18</b>
Creating a User Interface Model .....	<b>9-19</b>
Adding Dials & Gauges Blockset .....	<b>9-20</b>
Creating a Target Application .....	<b>9-21</b>
Running a Target Application with a User Interface Model ..	<b>9-23</b>

## **xPC Target Web Browser Interface**

# **10**

<b>Web Browser Interface</b> .....	<b>10-2</b>
Connecting the Web Interface Through TCP/IP .....	<b>10-2</b>
Connecting the Web Interface Through RS-232 .....	<b>10-3</b>
Using the Main Pane .....	<b>10-6</b>
Changing WWW Properties .....	<b>10-9</b>
Viewing Signals with a Web Browser .....	<b>10-10</b>
Viewing Parameters with a Web Browser .....	<b>10-11</b>
Changing Access Levels to the Web Browser .....	<b>10-11</b>

## **Interrupts Versus Polling**

# **11**

<b>Polling Mode</b> .....	<b>11-2</b>
xPC Target Kernel Polling Mode .....	<b>11-2</b>
Interrupt Mode .....	<b>11-2</b>
Polling Mode .....	<b>11-4</b>
Setting the Polling Mode .....	<b>11-6</b>
Restrictions Introduced by Polling Mode .....	<b>11-8</b>
Controlling the Target Application .....	<b>11-12</b>
Polling Mode Performance .....	<b>11-13</b>



**12**

**Introduction** . . . . . 12-2

    Simulink Demos Directory . . . . . 12-2

    Prerequisites . . . . . 12-3

    Steps to Incorporate Fortran in Simulink for xPC Target . . . . 12-3

**Step-by-Step Example of Fortran and xPC Target** . . . . . 12-5

    Creating an xPC Target Atmosphere Model for Fortran . . . . . 12-5

    Compiling Fortran Files . . . . . 12-7

    Creating a C-MEX Wrapper S-Function . . . . . 12-9

    Compiling and Linking the Wrapper S-Function . . . . . 12-9

    Validating the Fortran Code and Wrapper S-Function . . . . . 12-10

    Preparing the Model for the xPC Target Application Build . . 12-11

    Building and Running the xPC Target Application . . . . . 12-13

**Troubleshooting**

---

**13**

**General Troubleshooting Hints and Tips** . . . . . 13-2

**Installation, Configuration, and Test Troubleshooting** . . 13-6

**Advanced Troubleshooting** . . . . . 13-12

**Target PC Command-Line Interface Reference**

---

**14**

**Target PC Commands** . . . . . 14-2

    Target Object Methods . . . . . 14-3

    Target Object Property Commands . . . . . 14-3

    Scope Object Methods . . . . . 14-5

    Scope Object Property Commands . . . . . 14-8

    Aliasing with Variable Commands . . . . . 14-10

## Obsoleted xPC Target User Interfaces

### 15

<b>xPC Setup</b> .....	15-2
Environment Properties for Serial Communication .....	15-2
Environment Properties for Network Communication .....	15-4
Creating an xPC Target Boot Disk .....	15-7
Changing Environment Properties with a Graphical Interface	15-8
Saving and Loading the Environment Properties .....	15-10
<b>Control with xPC Target Remote Control Tool</b> .....	15-12
Signal Tracing with xPC Target Remote Control Tool .....	15-14
Signal Logging with xPC Target Remote Control Tool .....	15-25
Parameter Tuning with xPC Target Remote Control Tool ..	15-26

## Function Reference

### 16

<b>Functions — Categorical List</b> .....	16-2
Software Environment .....	16-2
GUI .....	16-3
Test .....	16-3
Target Objects .....	16-3
Scope Objects .....	16-6
File and File System Objects .....	16-7
<b>Functions — Alphabetical List</b> .....	16-10

## Index

# Target and Scope Objects

---

Before you can work with xPC Target target and scope objects, you should understand the concept of target and scope objects.

Target Objects (p. 1-2)

Description of target objects

Scope Objects (p. 1-3)

Description of scope objects

## Target Objects

xPC Target 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 PC with scope objects to collect signal data.

See Chapter 16, “Function Reference,” for a reference of the target functions.

### What Is a Target Object?

An understanding of the target object properties and methods will help you to control and test your application on the target PC.

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

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

To create a target object,

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

Target objects are of class `xpctarget.xpc`. A target object has associated properties and methods specific to that object.

## Scope Objects

xPC Target uses scope objects to represent scopes on the target PC. Use scope object functions to view and collect signal data.

See Chapter 16, “Function Reference,” for a reference of the scope functions.

### What Is a Scope Object?

xPC Target 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 PC.
- A scope that is outside a model is not a scope block. For example, if you create a scope with the `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. The topics in this section are

Refer to Chapter 1, “Target and Scope Objects,” for a description of how to use these objects, properties, and methods.

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

To create a scope object,

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

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

The following section describes scope object types.

## Scope Object Types

You can create scopes of type `target`, `host`, or `file`. Upon creation, xPC Target assigns the appropriate scope object data type for the scope type:

- `xpctarget.xpcsctg` for scopes of type `target`
- `xpctarget.xpcscho` for scopes of type `host`
- `xpctarget.xpcfs` for scopes of type `file`
- `xpctarget.xpsc` encompasses the object properties common to all the scope object data types. xPC Target creates this object if you create multiple scopes of different types for one model and combine those scopes, for example, into a scope vector.

Each scope object type has a group of object properties particular to that object type.

The `xpcsctg` scope object of type `target` has the following object properties:

- `Grid`
- `Mode`
- `YLimit`

The `xpcschost` scope object of type `host` has the following object properties:

- `Data`
- `StartTime`
- `Time`

The `xpcfz` scope object of type `file` has the following object properties:

- `AutoRestart`
- `Filename`
- `Mode`
- `StartTime`
- `WriteSize`

The `xpsc` scope object has the following object properties. The other scope objects have these properties in common:

- `Application`
- `Decimation`
- `NumPrePostSamples`
- `NumSamples`
- `ScopeId`
- `Status`
- `TriggerLevel`
- `TriggerMode`
- `TriggerSample`
- `TriggerScope`
- `TriggerSignal`
- `TriggerSlope`
- `Type`

See the scope object function `get (scope object)` on page 16-29 for a description of these object properties.





# 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 PC 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 topics.

- |                                      |   |
|--------------------------------------|---|
| Working with Target Objects (p. 2-2) | Use the MATLAB Command Window to change properties and use methods to control the target PC and your target application |
| Working with Scope Objects (p. 2-8)  | Use the MATLAB Command Window to change properties and use methods for signal logging and signal tracing                |

See Chapter 7, “Using the Target PC Command-Line Interface,” for a description of the target PC command-line interface.

# Working with Target Objects

This topic describes how to work with target objects using target object functions.

- “Creating Target Objects” on page 2-2
- “Deleting Target Objects” on page 2-3
- “Displaying Target Object Properties” on page 2-3
- “Setting Target Object Properties from the Host PC” on page 2-4
- “Resetting Target Application Parameters to Previous Values” on page 2-5
- “Getting the Value of a Target Object Property” on page 2-6
- “Using the Method Syntax with Target Objects” on page 2-7

See Chapter 16, “Function Reference,” for a reference of the target object functions.

## Creating Target Objects

To create a target object,

- Build a target application. xPC Target creates a target object during the build process.
- 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` as follows. In the MATLAB Command Window, type

```
tg = xpctarget.xpc
```

The resulting target object is `tg`.

- To create multiple target objects in your system, use the target object constructor function `xpc` (see `xpctarget.xpc` on page 16-117) as follows. For example, the following creates a target object connected to the host through an RS-232 connection. In the MATLAB window, type

```
tg1 = xpctarget.xpc('rs232', 'COM1', '115200')
```

The resulting target object is `tg1`.

To check a connection between a host and a target, use the target function `targetping`. For example,

```
tg.targetping
```

## Deleting Target Objects

To delete a target object, use the target object destructor function `delete`. In the MATLAB window, type

```
tg.delete
```

If there are any scopes still associated with the target, this function removes all those scope objects as well.

## 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 PC, 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 `get (target object)` on page 16-38

### Setting Target Object Properties from the Host PC

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

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

```
target_object.property_name = new_property_value
```

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

- 1 In the MATLAB window, type

```
tg.stoptime = 1000
```

- 2 Alternatively, you can type

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

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

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

Parameters are not target object properties. To change the frequency of the signal generator in the model `xpcosc`,

- 1 In the MATLAB window, type

```
setparam(tg,2,30)
```

The `setparam` method returns a result like the following:

```
ans =  
parIndexVec: 2  
OldValues: 20  
NewValues: 30
```

---

**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 PC. 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=setparam(tg,2,30)
```

The `setparam` method returns a result like the following:

```
pt =  
parIndexVec: 2  
OldValues: 20  
NewValues: 30
```

- 2 To reset to the previous values, type

```
setparam(tg,pt.parIndexVec,pt.OldValues)
```

```
ans =  
parIndexVec: 2  
OldValues: 30  
NewValues: 20
```

### Getting the Value of a Target Object Property

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

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

```
target_object.property_name
```

For example, to access the start time,

**1** In the MATLAB window, type

```
endrun = tg.stoptime
```

**2** 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`,

**1** In the MATLAB window, type

```
outputvalue= getsignal (tg,0)
```

**2** Alternatively, you could 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 scope of type `target` with a scope index of 1,

**1** In the MATLAB window, type

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

**2** Alternatively, you can type

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

# Working with Scope Objects

This topic describes how to work with scope objects using scope object functions.

- “Displaying Scope Object Properties for a Single Scope” on page 2-8
- “Displaying Scope Object Properties for All Scopes” on page 2-9
- “Setting the Value of a Scope Property” on page 2-9
- “Getting the Value of a Scope Property” on page 2-10
- “Using the Method Syntax with Scope Objects” on page 2-11
- “Acquiring Signal Data with Scopes of Type File” on page 2-12
- “Acquiring Gap-Free Data Using Two Scopes” on page 2-17
- “Acquiring Gap-Free Data Using Two Scopes” on page 2-17

See Chapter 16, “Function Reference,” for a reference of the scope object functions.

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

---

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

---



For a list of target object properties with a description, see the target function `get (target object)` on page 16-38.

## Displaying Scope Object Properties for All Scopes

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

- 1 In the MATLAB window, type  
`getscope(tg)` or `tg.getscope`

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

- 2 Alternatively, type  
`allscopes = getscope(tg)`

or type

```
allscopes = tg.getscope
```

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

For a list of target object properties with a description, see the target function `get (target object)` on page 16-38

## Setting the Value of a Scope Property

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

```
scope_object(index_vector).property_name = new_property_value
```

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

- 1 In the MATLAB window, type  
`sc1.triggermode = 'signal'`

- 2 Alternatively, you can type

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

or type

```
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, assuming you have a variable `sc12` for two scopes, 1 and 2, 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 xPC Target you can use either a function syntax or an object property syntax.

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

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

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

**1** In the MATLAB window, type

```
beginrun = sc1.starttime
```

**2** Alternatively, you can type

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

or type

```
sc1.get('starttime')
```

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

- `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, method names you must enter in full, and in lowercase. For example, to add signals to the first scope in a vector of all scopes,

1 In the MATLAB window, type

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

**2** Alternatively, you can type

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

### **Acquiring Signal Data with Scopes of Type File**

You can acquire signal data into a file on the target PC. To do so, you add a scope of type `file` to the application. After you build an application and download it to the target PC, you can add a scope of type `file` to that application.

For example, to add a scope of type `file` named `sc` to the application, and to add signal 4 to that scope,

**1** In the MATLAB window, type

```
sc=tg.addscope('file')
```

xPC Target creates a scope of type `file` for the application.

**2** Type

```
sc.addsignal(4)
```

xPC Target adds signal 4 to the scope of type `file`. 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 “Scope of Type File” on page 3-26 in Chapter 3, “Signals and Parameters,” for a description of signal tracing with scopes of type `file`.

## 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 `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.

- Pre-triggering — Starts to acquire data  $N$  moments before a trigger occurs
- Post-triggering — 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 2-14 — 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 2-17 — 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`
  - `sc1.NumPrePostSamples = P`

- For Scope 2, set the following values:

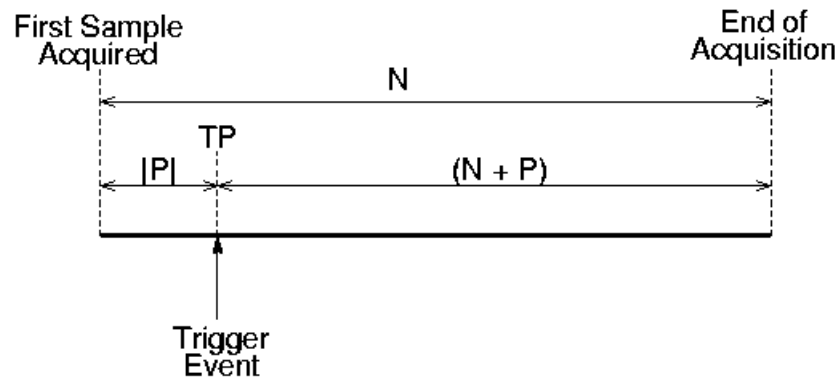
- `sc(2).ScopeId = 2`
- `sc(2).TriggerMode = 'Scope'`
- `sc(2).TriggerScope = 1`
- `sc(2).TriggerSample = range 0 to (N + P - 1)`

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 2-15 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.

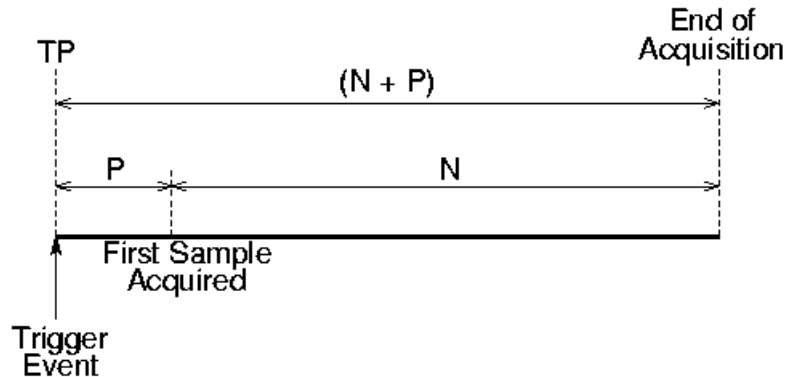
### Pre-triggering ( $P < 0$ )



### Pretriggering ( $P < 0$ )

“Posttriggering ( $P > 0$ )” on page 2-16, illustrates the behavior if  $P$ , the value of `NumPrePostSamples`, is positive. In this case, Scope 1 starts acquiring data  $|P|$  samples after TP occurs.

### Post-triggering ( $P > 0$ )



#### 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 section provides guidelines for setting up two scopes for gap-free data. This is functionality that you cannot achieve through pre- or posttriggering.

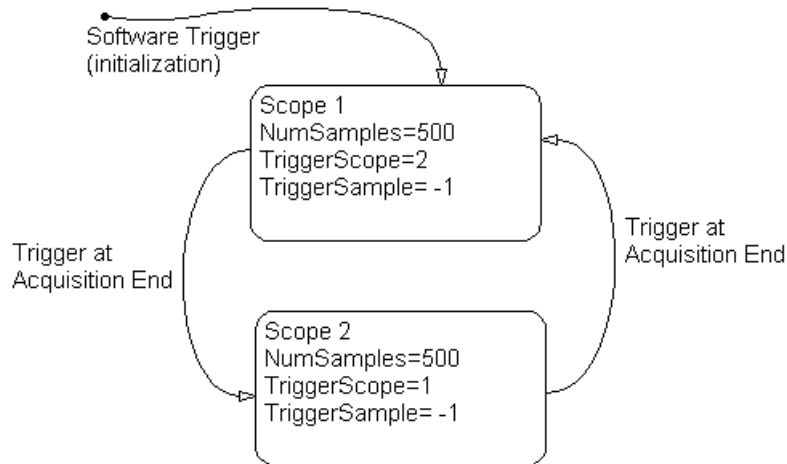
- Set the `TriggerSample` property for both scopes to `-1`. For example,

```
sc1.TriggerSample = -1
sc2.TriggerSample = -1
```
- Set the `TriggerScope` property for each scope so that each is triggered by the other. For example,

```
sc1.TriggerScope=2
sc2.TriggerScope=1
```
- Set the `NumSamples` property for each scope. For example,

```
sc1.NumSamples=500
sc2.NumSamples=500
```
- Set the `TriggerMode` property for one of the scopes to `'Software'`. You must do this to start the data acquisition. Otherwise, each scope waits for the other to finish acquiring data, and never starts. In “Acquisition of Gap-Free Data” on page 2-18, 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`.
- Both the scopes receive exactly the same signals, in other words, the signals you want to retrieve.

“Acquisition of Gap-Free Data” on page 2-18, illustrates how the scopes trigger one another.



### Acquisition of Gap-Free Data

The following code is a typical example of how you can retrieve gap-free data. You can type this code into an m file and run that file for a downloaded target application. This example assumes that the communication speed and number of samples are fast enough to acquire the full data set before the next acquisition cycle is due to start. You can also use more than two scopes to implement a triple- or quadruple-buffering scheme instead of the double-buffering one illustrated here.

```
% Assumes that model is built and loaded on target.
tg = xpctarget.xpc;
sc = tg.addscope('target', [1 2]);
addsignal(sc,[0 1]);
% [0 1] are the signals of interest; add to both
% Default value for TriggerSample is 0, need to change it.
set(sc, 'NumSamples', 500, 'TriggerSample', -1)
set(sc, 'TriggerMode', 'Scope');
sc(1).TriggerScope = 2;
sc(2).TriggerScope = 1;
start(sc);
start(tg);
```

```
sc(1).trigger;
% Start things off by triggering scope 1
data = zeros(0, 2);
t     = [];
scNum = 1;
% We will look at scope 1 first
% Use some appropriate condition instead of an infinite loop
while(1)
    % loop until the scope has finished
    while ~strcmp(sc(scNum).Status, 'Finished'), end
        data(end + 1 : end + 500, :) = sc(scNum).Data;
        t(   end + 1 : end + 500)     = sc(scNum).Time;
        start(sc(scNum));
    % Restart the scope
    scNum = 3 - scNum;
% Switch to the next scope
end
```



# 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. xPC Target includes command-line and graphical user interfaces to complete these tasks. This chapter includes the following sections:

Signal Monitoring with MATLAB  
(p. 3-5)

Acquire signal data while running a target application without time information

Signal Tracing (p. 3-7)

Acquire and visualize signals while running a target application in real time

Signal Logging (p. 3-28)

Acquire signal data while running a target application, and after the run, transfer the data to the host PC for analysis

Parameter Tuning (p. 3-35)

Change parameters in your target application while it is running in real time

This chapter describes xPC Target interactions using a number of interfaces, including xPC Target Explorer. For additional information on using xPC Target Explorer, see Chapter 4, “xPC Target Explorer.”

# Monitoring Signals

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 PC.

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

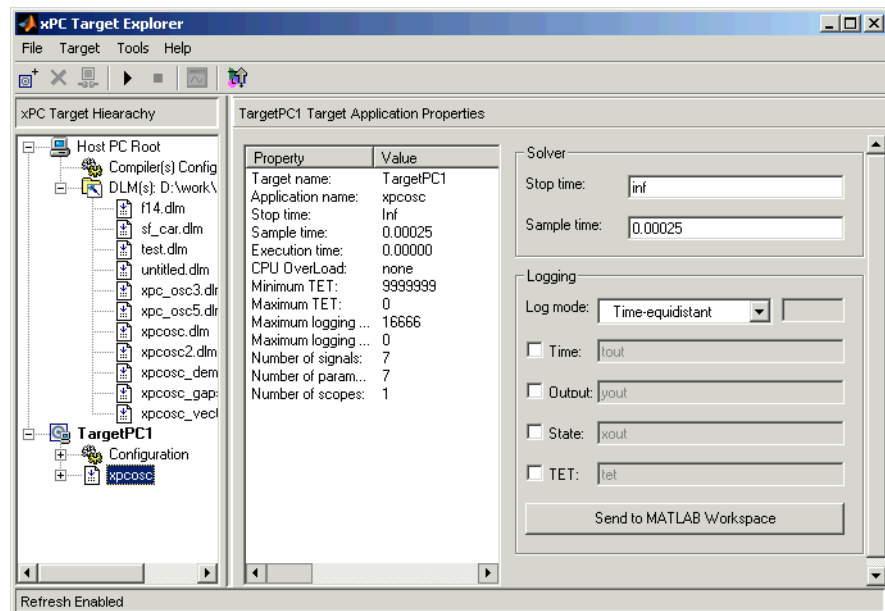
This section has the following topics:

- “Signal Monitoring with xPC Target Explorer” on page 3-2
- “Signal Monitoring with MATLAB” on page 3-5

## Signal Monitoring with xPC Target Explorer

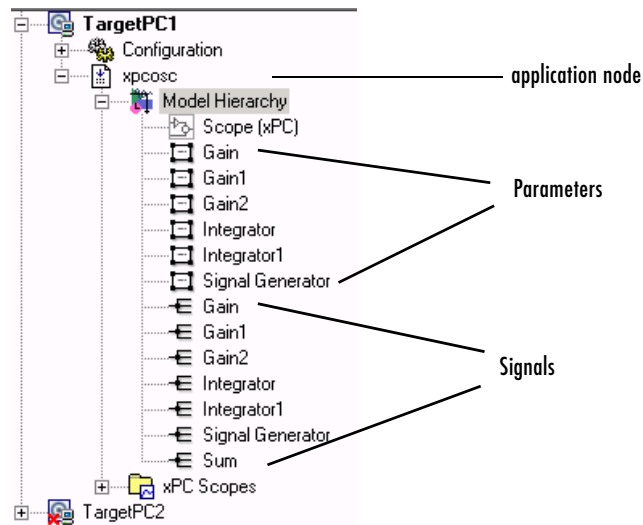
This procedure uses the model `xpcosc.mdl` as an example, and assumes you created and downloaded the target application to the target PC. For meaningful values, the target application should be running.




- 1 In xPC Target Explorer, go to the node of the running target application in which you are interested. For example, `xpcosc`.



- 2 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.

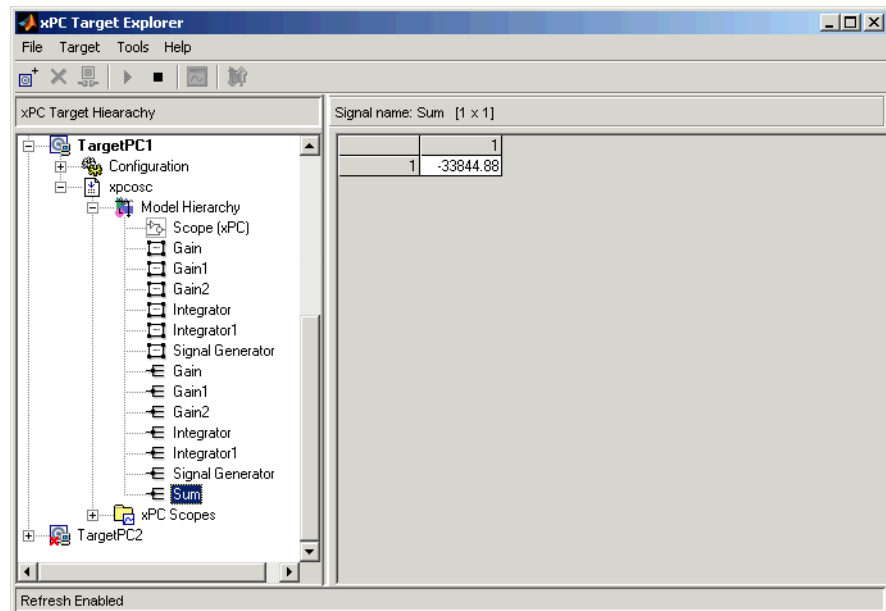


The Model Hierarchy node can have up to three nodes: subsystems (  ) (including their signals and parameters), parameters (  ), and signals (  ). Only xPC Target tunable parameters and signals of the application, as represented in the Simulink model, appear in the Model Hierarchy node. This example currently has only parameters and signals.

- 3 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-hand pane. For example,





## Signal Monitoring with MATLAB

This procedure uses the model `xpc_osc3.mdl` as an example, and assumes you created and downloaded the target application to the target PC.

- 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.

```
ShowSignals = On
```

Signals =	INDEX	VALUE	BLOCK NAME
	0	0.000000	Transfer Fcn
	1	0.000000	Signal Generator

**2** To get the value of a signal, in the MATLAB Command Window, type  
`tg.getsignal(1)`

MATLAB displays the value of signal 1.

```
ans=  
    3.731
```

See also “Signal Tracing with MATLAB” on page 3-17.

## Signal Tracing

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

- “Signal Tracing with xPC Target Explorer” on page 3-7 — Use the xPC Target Explorer to create and run scopes that are displayed on the host PC.
- “Signal Tracing with MATLAB” on page 3-17 — Use the MATLAB Command Window to create scopes and scope objects.
- “Signal Tracing with xPC Target Scope Blocks” on page 3-25 — Use scopes of type host to trace signal data triggered by an event while your target application is running.
- “Signal Tracing with a Web Browser” on page 3-26 — Use Microsoft Explorer or Netscape Navigator to view signals.

Signal tracing differs from signal logging. With signal logging you can only look at signals after a run is finished, and the entire log of the run is available. For information on signal logging, see “Signal Logging” on page 3-28.

### Signal Tracing with xPC Target Explorer

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

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

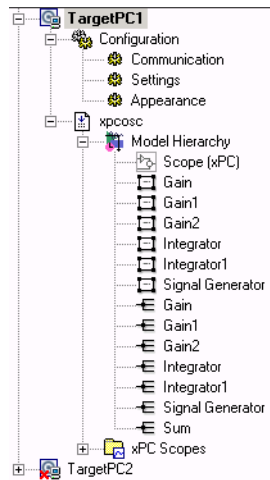
- “Creating Scopes” on page 3-8 — Create scopes on the host PC and target PC to visualize the data.
- “Adding Signals to Scopes” on page 3-11 — Add signals to the scopes and start the scopes.
- “Stopping Scopes” on page 3-15 — Stop the scopes.

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

## Creating Scopes

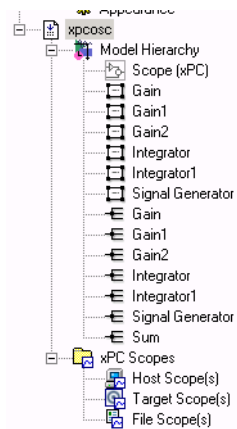
- 1 In xPC Target Explorer, right-click the downloaded target application node. For example, xpcosc.
- 2 Select **Start**.
- 3 To get the list of signals in the target application, expand the Model Hierarchy node under the target application.

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



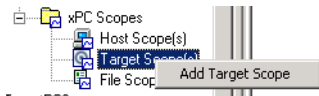
- 4 To get the list of scope types you can have in the target application, expand the xPC Scopes node below the application node.

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



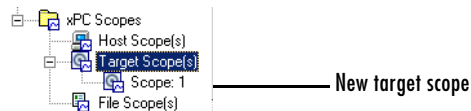
- 5 To create a scope to display on the target PC, right-click Target Scopes from the xPC Scopes node.

A dialog appears. This dialog lists the operations you can perform on target PC scopes. For example, within the current context, you can create a target PC scope.



- 6 Select **Add Target Scope**.

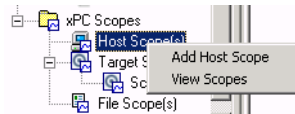
A scope node appears under Target Scopes. In this example, the new scope is labeled as Scope 1.



You can add other scopes, including those of type host and file.

- 7 To create a scope to display on the host PC, right-click **Host Scopes** from the **xPC Scopes** node.

A dialog appears. This dialog lists the operations you can perform on host PC scopes. For example, within the current context, you can create a host PC scope.

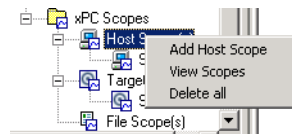


- 8 Select **Add Host Scope**.

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

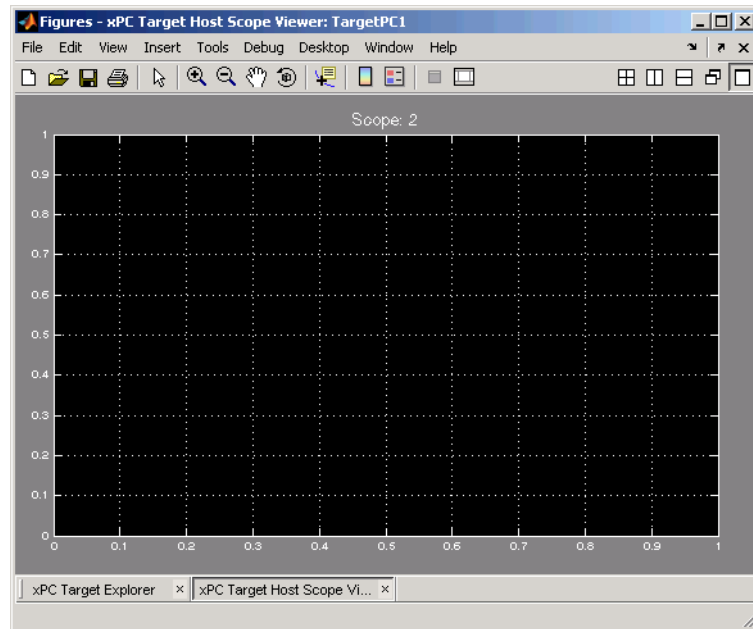
- 9 To visualize the host scope on the host PC, right-click **Host Scopes** from the **xPC Scopes** node.

A dialog appears.



- 10 Select **View Scopes**.

The **xPC Target Host Scope Viewer** appears. By default, the viewer is docked in the same window as the **xPC Target Explorer** window. Note that the signals you add to the scope will appear at the top right of the viewer.



- 11 To list the properties of the scope object Scope 2, select the xPC Target Explorer tab to return to that window and left-click Scope 2.

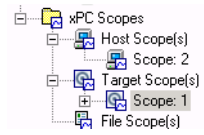
The scope properties are shown in the rightmost pane.

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

### Adding Signals to Scopes

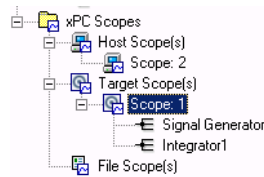
- 1 In the xPC Target Explorer window, add signals to the target PC scope, Scope 1. For example, to add signals Integrator1 and Signal Generator, right-click each signal and select **Add to Scopes**. From the **Add to Scopes** list, select Scope 1.

The Scope 1 node is shown with a plus sign.

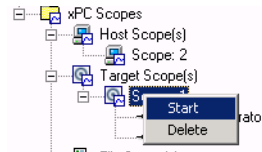


- 2 Expand the Scope 1 node.

The Scope 1 signals are displayed.

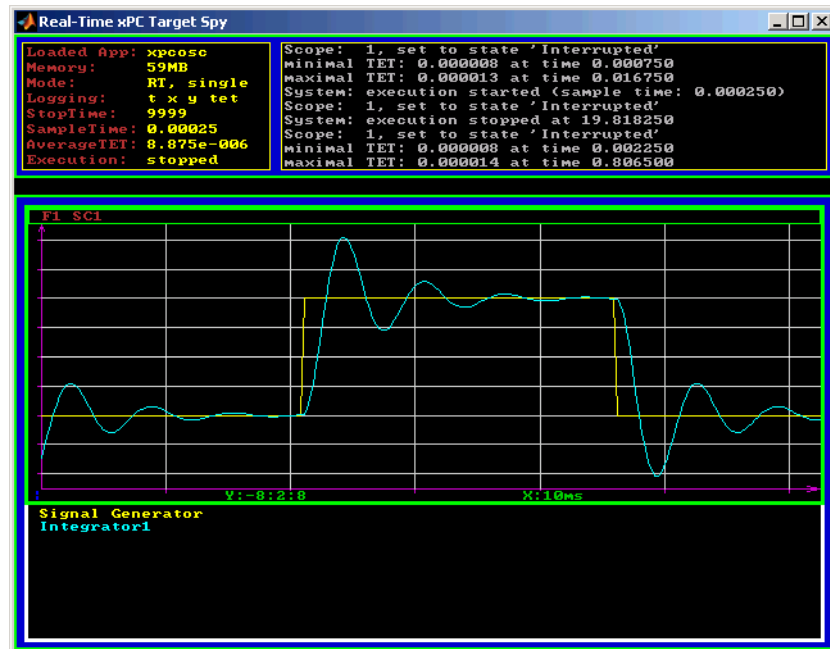


- 3 Start the scope. For example, to start Scope 1, right-click Scope 1 and select **Start**.



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.



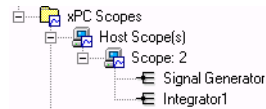


- 4 Add signals to the host PC scope. For example, to add signals Integrator1 and Signal Generator, right-click each signal and select **Add to Scopes**. From the **Add to Scopes** list, select Scope 2.

The Scope 2 node is shown with a plus sign.

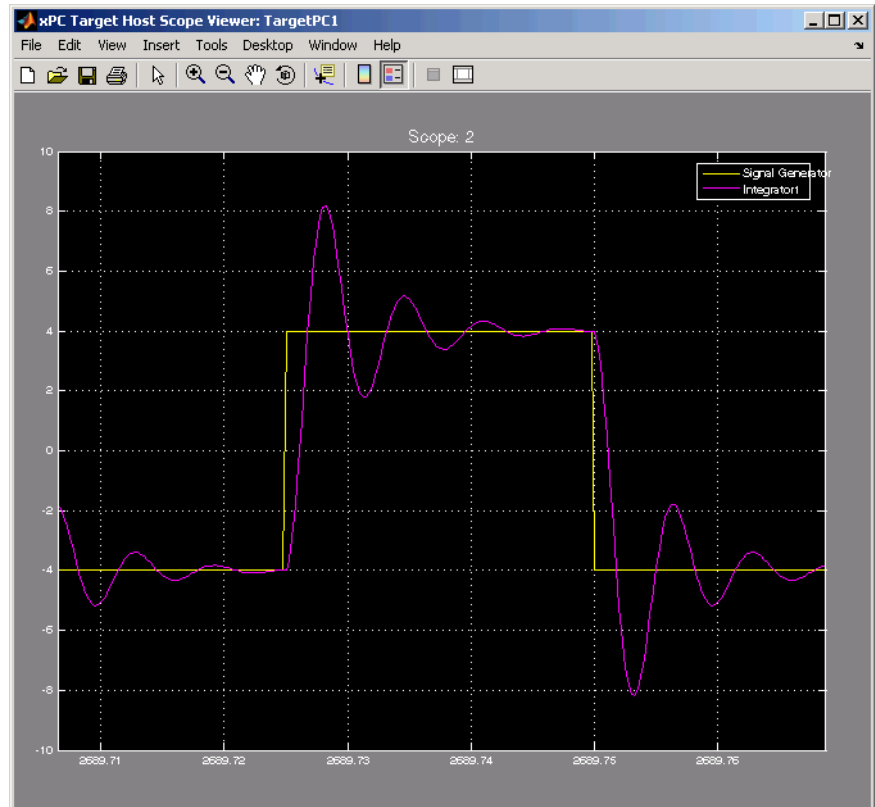
- 5 Expand the Scope 2 node.

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.



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 Scope 1 and select **Stop**.

The signals shown on the target PC stop updating while the target application continues running, and the target PC 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 PC stops running, and the target PC displays the following messages:

```
System: execution stopped
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.

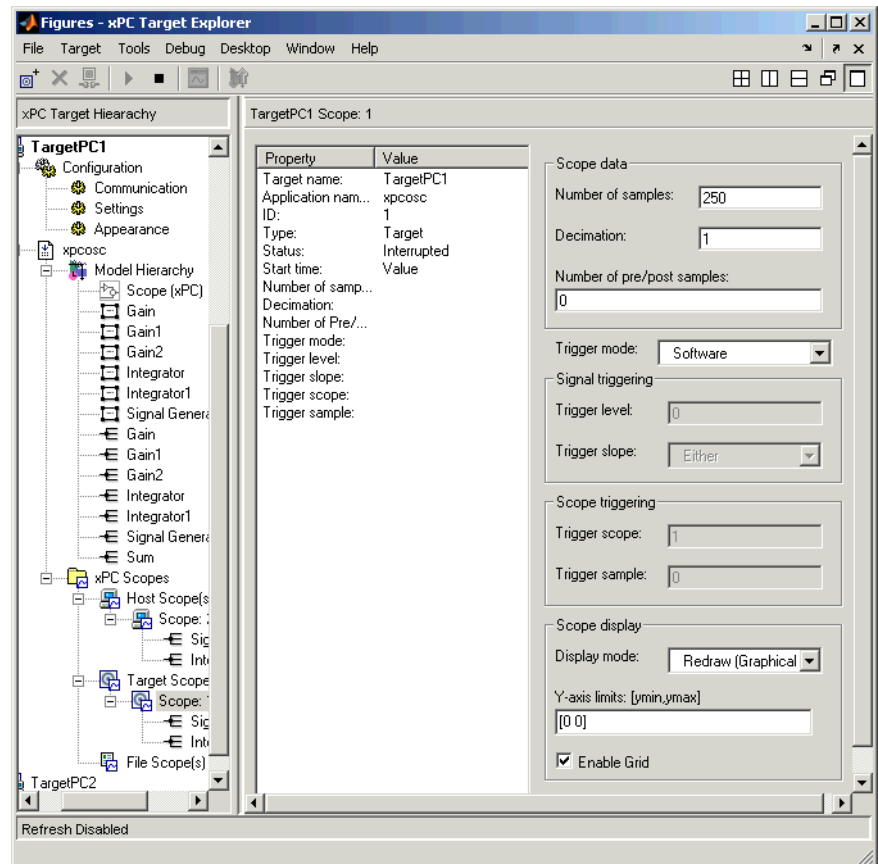
## Software Triggering Scopes

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 3-8) and that you have added signals to that scope (“Adding Signals to Scopes” on page 3-11).

- 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.



- 3 Start the scope and target application.
- 4 Right-click on the scope to be triggered. For example, select Scope 1.
- 5 Select Trigger.

### 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 3-8 and “Adding Signals to Scopes” on page 3-11). 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 MATLAB.

## Signal Tracing with MATLAB

Creating a scope object allows you to select and view signals using the scope methods. This section describes how to signal trace using xPC Target functions instead of using the xPC Target graphical user interface.

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 MATLAB and Scopes of Type Target” on page 3-18 for a description of signal tracing with target scopes)

- Scopes of type `file` (see “Signal Tracing with MATLAB and Scopes of Type File” on page 3-21)

Note, stop the scope before adding or removing signals from the scope.

### Signal Tracing with MATLAB and Scopes of Type Target

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 either

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

The target PC 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
              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 MATLAB” on page 3-5.

- 3** Create a scope to display on the target PC. 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 StartTime, Time, and Data are not accessible with a scope of type target.

```
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 PC 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 sc.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 PC stop updating while the target application continues running, and the target PC 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 PC stops running, and the target PC displays the following messages.

```
System: execution stopped
minimal TET: 0.000023 at time 1313.789000
maximal TET: 0.000034 at time 407.956000
```



## Signal Tracing with MATLAB and Scopes of Type File

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 scopes of type file.

---

**Note** The signal data file can quickly increase in size. You should examine the file size between runs to gauge the growth rate for 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 scopes of type file. Type

```
tg=xpctarget.xpc
```

- 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
            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 MATLAB” on page 3-5.

- 3** Start running your target application. Type

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

The target PC displays the following message:

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

- 4** Create a scope to be displayed on the target PC. 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 StartTime, Time, and Data are not accessible with a scope of type target.

```
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
Signals              = no Signals defined
StartTime            = -1.000000
FileName             = unset
```

```

Mode                = Lazy
WriteSize           = 512
AutoRestart         = off

```

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 ([0,1]) or addsignal(sc2,[0,1])
```

The target PC displays the following messages.

```

Scope: 2, signal 0 added
Scope: 2, signal 1 added

```

After you add signals to a scope object, the scope of type `file` 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 target PC displays the following message.

```
FileSys:File c:\sc2Integ.dat opened
```

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 scope of type `file`. 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     = 0
TriggerLevel      = 0.000000
TriggerSlope      = Either
StartTime         = NaN
Signals           = 0 : Integrator1
                  1 : Signal Generator

StartTime         = NaN
FileName          = c:\sc2Integ.dat
Mode              = Lazy
WriteSize         = 512
AutoRestart       = off
```

**8** Stop the scope. Type

```
-sc2 or sc2.stop or stop(sc2)
```

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

```
FileSys:File c:\sc2Integ.data closed
Scope: 2, set to state 'Interrupted'
```

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

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

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

```
System: execution stopped
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 xPC Target scope of type file creates, use the xPC Target file system object (`xpctarget.fs`) from the host PC MATLAB window. To view or examine the signal data, you can use the `readxpcfile` utility in conjunction with the `plot` function. For further details on the `xpctarget.fs` file system object and the `readxpcfile` utility, see Chapter 8, “Working with Target PC Files and File Systems.”

---

## Signal Tracing with xPC Target Scope Blocks

Use scopes of type host 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 10 scopes of each scope type for a maximum of 30 scopes.

---

### Scope of Type Host

For a scope of type host, 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.

### Scope of Type Target

For a scope of type target, logged data (`sc.Data`, `sc.Time`, and `sc.StartTime`) is not accessible over the command-line interface on the host PC. 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 scope of type target, and then try to get the logged data by typing `sc.Data`, you will 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 PC while the data is displayed on the target PC, you need to define a second scope object with type host. Then you need to synchronize the acquisition of the two scope objects by setting **TriggerMode** for the second scope to 'Scope'.

### Scope of Type File

For a scope of type `file`, 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. The memory buffer writes data to the file at the end of the session.

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.

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 a Web Browser

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

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

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

The browser loads the **Scopes List** pane into the right frame.

- 2 Click the **Add Scope** button.

A scope of type `target` is created and displayed on the target PC. The **Scopes** pane displays a list of all the scopes present. You can add a new scope, remove existing scopes, and control all aspects of a scope from this page.

To create a scope of type `host`, 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 necessary, 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 variable (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

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 PC for analysis. 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 ensure that signals are sampled at their appropriate sample rates.

---

**Note** xPC Target does not support logging data with decimation.

---

This section includes the following topics:

- “Signal Logging with xPC Target Explorer” on page 3-28 — Use an xPC Target Scope block in your Simulink model to log signal data triggered by an event.
- “Signal Logging with MATLAB” on page 3-30 — Use Outport blocks in your Simulink model to log data to a target object in the MATLAB workspace.
- “Signal Logging with a Web Browser” on page 3-34 — Use Microsoft Internet Explorer or Netscape Navigator to log data to a text file.

## 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 the model `xpc_osc4.mdl` as an example and assumes you have created a target application and downloaded it to the target PC. See “xPC Target Application” in Chapter 3 in the xPC Target Getting Started documentation.

---

**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.

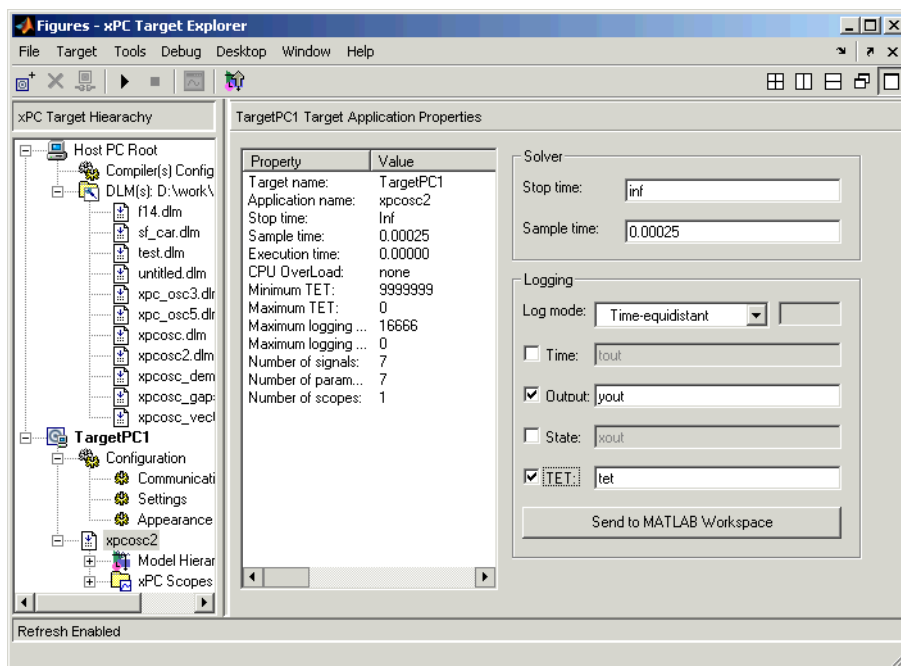
---



- 1 In xPC Target Explorer, select the down-loaded target application node. For example, `xpc_osc4`.

The right pane displays the target application properties dialog for `xpc_osc4`.

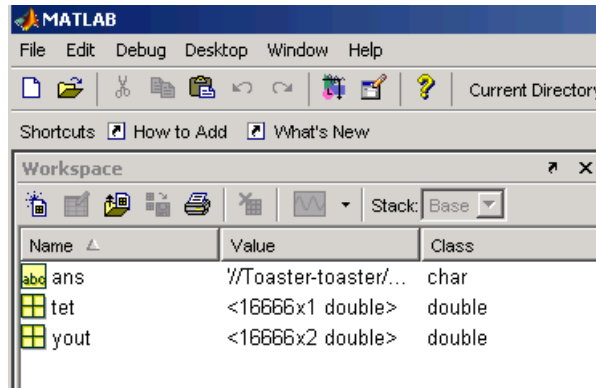
- 2 In the **Logging** pane, select the box of the signals you are interested in logging. For example, select **Output** and **TET**.



- 3 Start the target application. For example, in the **Target Hierarchy** pane, select the `xpc_osc4` target application, then select **Start**.
- 4 Stop the target application. For example, in the **Target Hierarchy** pane, select the `xpc_osc4` target application, then select **Stop**.

- 5 Send the selected logged data to the MATLAB workspace. In the target application properties dialog for `xpc_osc4`, go to the **Logging** pane and press the **Send to MATLAB Workspace** button.

In the MATLAB desktop, the Workspace pane displays the logged data.



You can examine or otherwise manipulate the data.

## Signal Logging with MATLAB

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 Blocks” in Chapter 3 of the xPC Target Getting Started documentation.

**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” in Chapter 3 of the xPC Target Getting Started documentation.

All scopes copy the last N samples from the log buffer to the target object logs (`tg.TimeLog`, `tg.OutputLog`, `tg.State`, and `Log`, `tg.TETLog`). xPC Target 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.

- 1 In the MATLAB window, 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`.

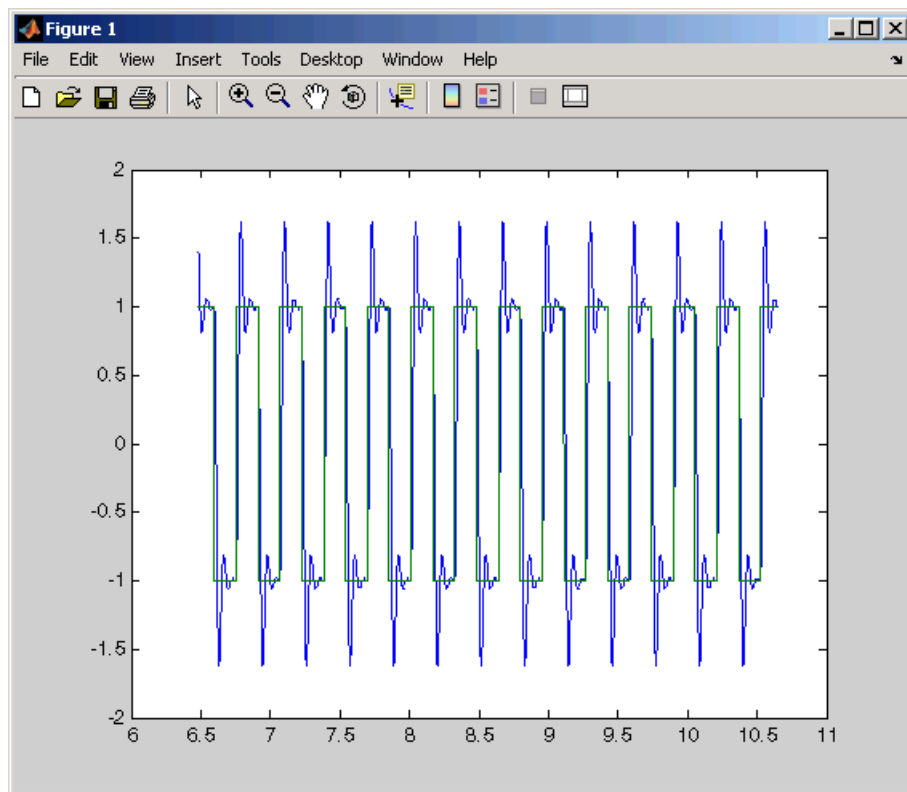
The outputs are the signals connected to Simulink Outport blocks. The model `xpcosc.mdl` has just one Outport block, labeled **1**, and there are two states. This Outport block shows the signals leaving the blocks labeled `Integrator1` and `Signal Generator`.

- 2 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 PC from the target application on the target PC. If you want to upload part of the logs, see the target object method `getlog` on page 16-45.

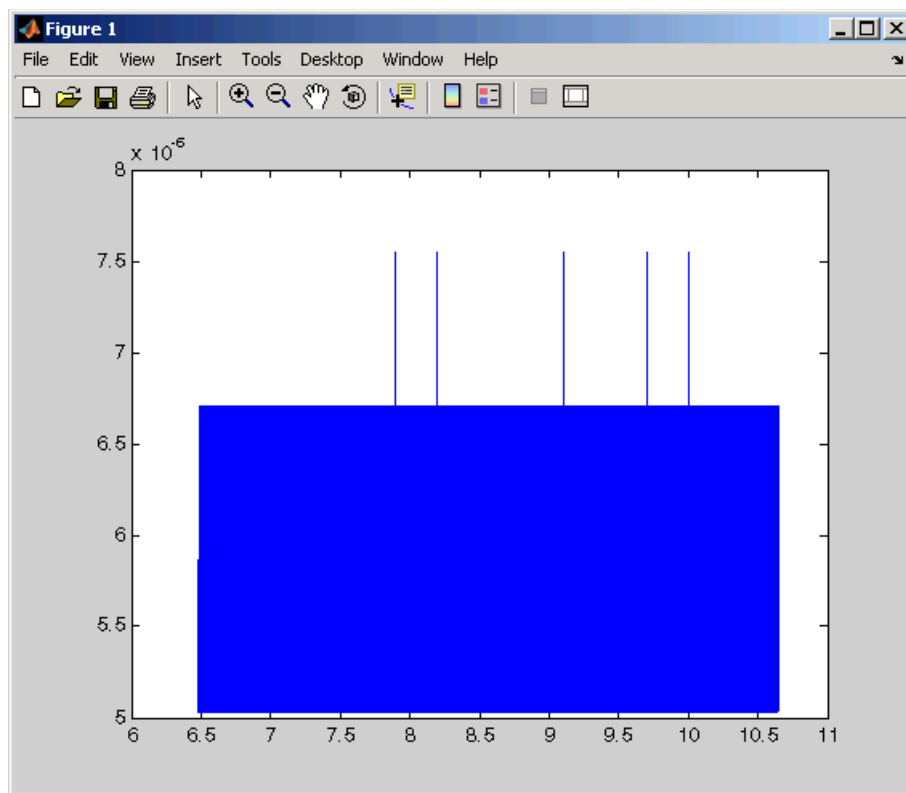
The plots shown below are the result of a real-time execution. To compare this plot with a plot for a non-real-time simulation, see “Simulating the Model from MATLAB” in Chapter 3 of the xPC Target Getting Started documentation.



- 3** In the MATLAB window, type  
`plot(tg.TimeLog,tg.TETLog)`

Values for the task execution time (TET) log are uploaded to the host PC from the target PC. If you want to upload part of the logs, see the target object method `getlog` on page 16-45.

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.

4 In the MATLAB window, type either

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

MATLAB 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.

### 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 MATLAB using the `csvread` function.

This section of the Web browser interfaces 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 outport 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 Outport Blocks” in Chapter 3 in the xPC Target Getting Started documentation.

**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 Chapter 3 in the xPC Target Getting Started documentation.

## Parameter Tuning

xPC Target lets you change parameters in your target application while it is running in real time. This section includes the following topics:

- “Parameter Tuning with xPC Target Explorer” on page 3-35 — Use the xPC Target Explorer to change block parameters in your target application.
- “Parameter Tuning with MATLAB” on page 3-38 — Use the MATLAB Command Window and target objects in your MATLAB workspace to change target application parameters.
- “Parameter Tuning with Simulink External Mode” on page 3-40 — Connect your Simulink model to your target application, and change target application parameters by changing Simulink block parameters.
- “Parameter Tuning with a Web Browser” on page 3-42 — Connect your target application to a Web browser with the target application running on a target PC connected to a network.

### Parameter Tuning with xPC Target Explorer

xPC Target lets you change parameters in your target application while it is running in real time. With these functions you do not need to set Simulink to external mode, and you do not need to connect Simulink with the target application.

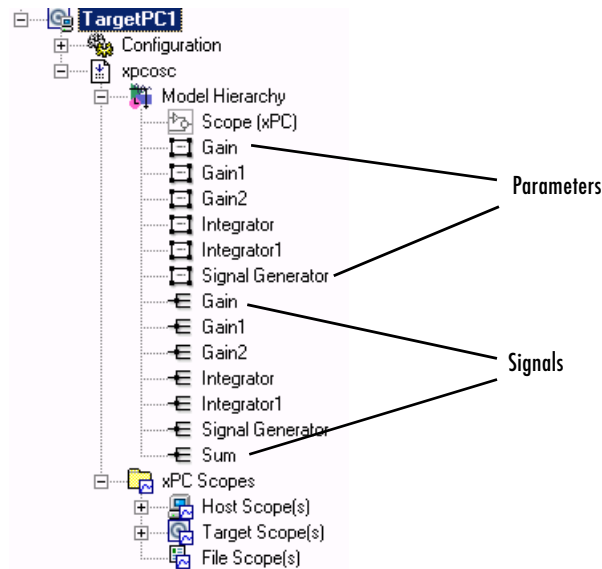
You can download parameters to the target application while it is running or between runs. This feature lets you change parameters in your target application without rebuilding the Simulink model.

After you download a target application to the target PC, 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.



Each signal in the signals section has a corresponding entry in the parameters section. To change the value of a signal, you need to select the corresponding parameter entry. Note that if a parameter is not tunable, it does not appear in the parameter area.

- 4 Select the parameter of the signal you want to edit. For example, select Signal Generator.

The right pane displays the Block Parameters dialog for Signal Generator. There are two parameters, **Amplitude** and **Frequency**, for this block.

- 5 Change the frequency. For example, in the Signal Generator block parameters dialog, select Frequency.

The current value of the **Frequency** parameter is displayed.



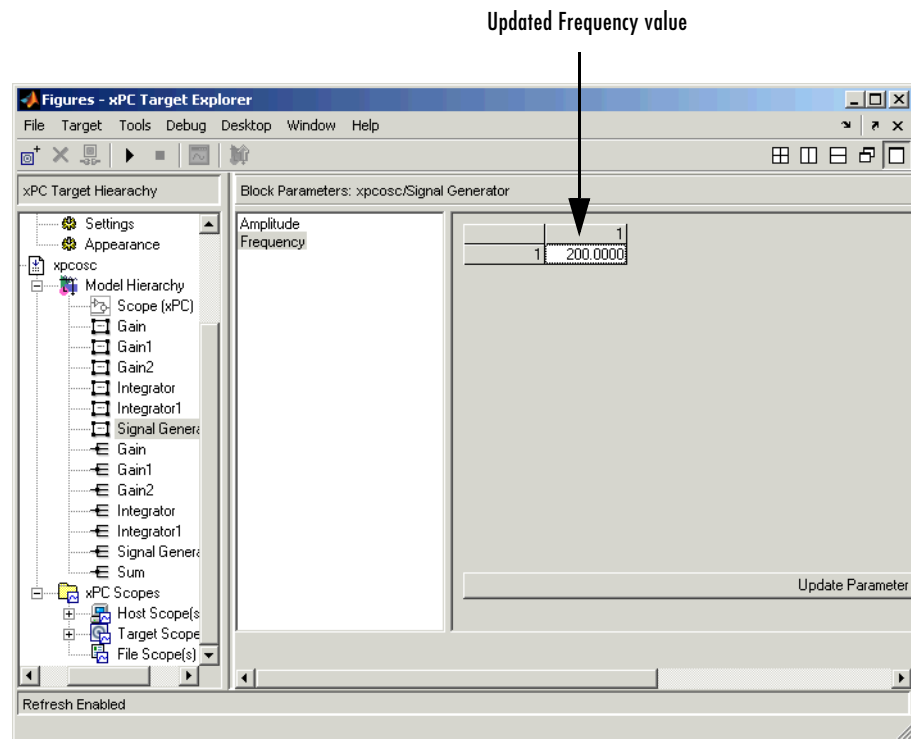
- 6 Double-click the box that contains the frequency value.

The box becomes editable.

- 7 Enter a new value for the frequency. For example, 200.

- 8 Press the **Enter** key.

The box is updated and the Update Parameter button becomes active.



The plot frame then updates the signals after running the simulation with the new parameter value.

- 9 Stop the scope. For example, to stop Scope 1, right-click Scope 1 and select **Stop**.

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

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

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

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

```
System: execution stopped  
minimal TET: 0.0000006 at time 0.001250  
maximal TET: 0.0000013 at time 8.557000
```

## Parameter Tuning with MATLAB

You use the MATLAB functions to change block parameters. With these functions you do not need to set Simulink to external mode, and you do not need to connect Simulink 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 PC, 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:

- 1 In the MATLAB window, type  

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

The target PC 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'
```

and then type

```
tg
```

The MATLAB window displays a list of properties for the target object.

```
ShowParameters = on
```

```
Parameters =
```

INDEX	VALUE	TYPE	SIZE	PARAMETER NAME	BLOCK NAME
0	0	DOUBLE	Scalar	Initial Condition	Integrator1
1	4	DOUBLE	Scalar	Amplitude	Signal Generator
2	20	DOUBLE	Scalar	Frequency	Signal Generator
3	1000000	DOUBLE	Scalar	Gain	Gain
4	0	DOUBLE	Scalar	Initial Condition	Integrator
5	400	DOUBLE	Scalar	Gain	Gain1
6	1000000	DOUBLE	Scalar	Gain	Gain2

**3** Change the gain. For example, to change the Gain1 block, type either

```
tg.setparam(5,800) or setparam(tg,5,800)
```

As soon as you change parameters, the changed parameters in the target object are downloaded to the target application. The target PC displays the following message:

```
ans =  
  
parIndexVec: 5  
OldValues: 100  
NewValues: 800
```

The plot frame then updates the signals after running the simulation with the new parameters.

- 4 Stop the target application. In the MATLAB window, type  
-tg or tg.stop or stop(tg)

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

```
System: execution stopped  
minimal TET: 0.000023 at time 1313.789000  
maximal TET: 0.000034 at time 407.956000
```

## 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 Simulink 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, Simulink 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 PC, 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 to your target application. This downloading guarantees the consistency of the parameters between the host model and the target application.

The target PC displays the following message, where # is the actual number of tunable parameters in your model:

```
ExtM: Updating # parameters
```

- 3 From the **Simulation** menu, click **Start Real-Time Code**, or, in the MATLAB window, type

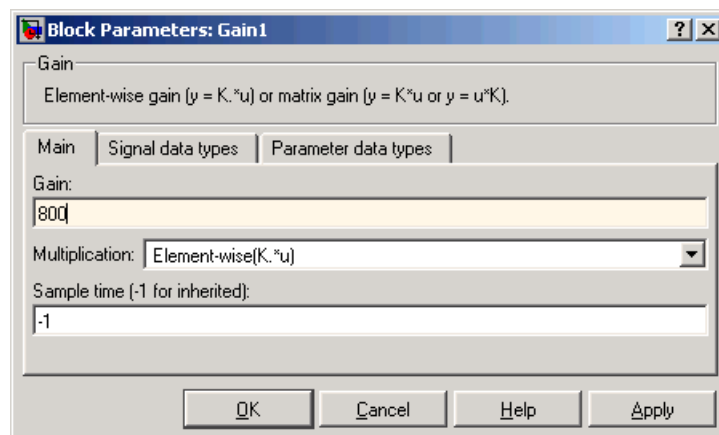
```
+tg or tg.start or start(tg)
```

The target application begins running on the target PC, and the target PC 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.



- In the **Gain** text box, enter 800 and click **OK**.

As soon as you change a model parameter and click **OK** or the **Apply** button on the **Block Parameters: Gain1** dialog box, all the changed parameters in the model are downloaded to the target application, as shown below.

Loaded App: <b>xpcosc</b>	System: COM1 detected, BaudRate: 115200
Memory: <b>124MB</b>	System: download started...
Mode: <b>RT, single</b>	System: download finished
Logging: <b>t x y tet</b>	System: initializing application...
StopTime: <b>1e+008</b>	System: initializing application finished
SampleTime: <b>0.00025</b>	ExtM: updating 7 parameters
AverageTET: <b>5.126e-006</b>	System: execution started (sample time: 0.000250)
Execution: <b>74.40 s</b>	ExtM: updating parameter

- 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 Simulink works regardless of whether the target application is running or not.

- From the **Simulation** menu, click **Stop real-time code**, or, in the MATLAB window, type either

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

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

```
System: execution stopped
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 PC you can use the **Parameters** page to change parameters in your target application while it is running in real time:

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

The browser loads the **Parameter List** pane into the right frame.

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

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

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

The new parameter values are uploaded to the target application.

## **Saving and Reloading Application Parameters with MATLAB**

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 PC. 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 a target application downloaded on the target PC.
- You have parameters you would like to save for reuse. See
  - “Parameter Tuning with MATLAB” on page 3-38
  - “Parameter Tuning with Simulink External Mode” on page 3-40
  - “Parameter Tuning with a Web Browser” on page 3-42

### **Saving the Current Set of Target Application Parameters**

To save a set 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')
```

xPC Target creates a file named `xpcosc4_param1` in the current directory of the target PC, 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 3-44. 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 3-45.

### **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 3-44).

- 1** From the collection of parameter value files on the target PC, 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')
```

xPC Target loads the parameter values into the target application.



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 3-45.

### **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 3-44).

- 1** Ensure that the target application is stopped. 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.



# xPC Target Explorer

---

This chapter describes some configuration and use features of xPC Target Explorer.

Introduction (p. 4-2)	Read about the operations you can perform through xPC Target Explorer
Specifying Target PC Configurations (p. 4-4)	Define xPC Target PC configurations
Preparing Target PCs for xPC Target System Use (p. 4-8)	Configure target PCs with predefined configurations
Downloading and Running Target Applications (p. 4-10)	Download and run a target application on target PC
Control with xPC Target Explorer (p. 4-15)	Manipulate target application signals and parameters
Menu Bar and Toolbar Contents and Shortcut Keys (p. 4-17)	Refer to this section for the menu bar and toolbar contents

## Introduction

The xPC Target Explorer is a graphical user interface for xPC Target. 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 PC for xPC Target
- Add and configure target PCs for xPC Target
- Create boot disks for particular target PCs
- Connect the target PCs for your xPC Target system to the host PC
- Download a prebuilt target application, DLM, to a target PC
- Start and stop the application that has been downloaded to the target
- Add scopes of type host, target, and file to the downloaded target application
- Add and remove signals to the xPC Target scopes
- 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 machine.

This chapter describes some of the xPC Target Explorer features. For more detailed information, see

- Chapter 2, “Installation and Configuration” and Chapter 3, “Basic Tutorial” in the xPC Target Getting Started Guide documentation
- Chapter 3, “Signals and Parameters” of the current guide

### Before You Start

xPC Target Explorer assumes the following:

- At least one of the supported C compilers is installed.
- You have existing prebuilt target applications (see “Simulink Model” in Chapter 3 of the xPC Target Getting Started documentation for a description of how to create a Simulink model, and generate a target application from that Simulink model).

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 this chapter describe procedures using mouse operations.

## Specifying Target PC Configurations

- “Configuring Environment Parameters for Target PCs” on page 4-4 — Describes how to configure extra target PC environment variables
- “Creating a Target PC Boot Disk” on page 4-5 — Describes how to create a boot disk for the target PC just configured

### Configuring Environment Parameters for Target PCs

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

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

---

**Note** In general, the default values of these parameters are sufficient for you to use xPC Target.

---

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

A Configuration node appears. Under this are nodes for Communication, Settings, and Appearance. The parameters for the target PC 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 application automatically determines the amount of memory up to 64 MB.
- The amount of RAM, in MB, installed on the target PC.

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

- 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 PC for the target application. The remaining memory is used by the kernel and by the heap for data logging.
- 5 From the **CAN library** list, select None, 200 ISA, 527 ISA, 1000 PCI, 1000 MB PCI, or PC104.
- 6 In the **xPC Target Hierarchy**, select Appearance.  
  
The **Appearance Component** pane appears to the right.
- 7 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 PC displays information as text. To use all the features of the target scope, you also need to install a keyboard and mouse on the target PC.
- 8 Set the **Target scope** property to Enabled.
- 9 **Target mouse** allows you to disable or enable mouse support on the target PC. From the **Target mouse** list, select None, PS2, RS232 COM1, or RS232 COM2.

You are now ready to create an xPC Target boot disk for the target PCs. See “Creating a Target PC Boot Disk” on page 4-5.

## Creating a Target PC Boot Disk

When you are satisfied with the host PC and target PC configurations, you can create an xPC Target boot disk. An xPC Target boot disk includes the xPC Target kernel specific for either serial or network communication. Each unique host PC/target PC configuration combination requires its own target boot disk.

You can create one of three types of boot disk per configuration combination:

- **Boot floppy** — Allows you to boot a target PC from a 3.5 inch disk. You can then download a target application from the host PC to the target PC. This is the most basic of the target boot disks. The following procedure describes how to create Boot Floppy disk.
- **DosLoader** — Allows you to boot a target PC from a device other than a 3.5 inch disk, such as a hard disk or flash memory. You can then download a

target application from the host PC to the target PC. This mode requires you to have your own copy of DOS on a 3.5 inch disk. See Chapter 5, “Embedded Option,” of the xPC Target User’s documentation for a description of how to create a DosLoader disk.

- **StandAlone** — Bundles the kernel and target application into one entity that you can copy onto a 3.5 inch disk or alternate device. This allows the target PC to run as a stand-alone PC with the target application already loaded. See Chapter 5, “Embedded Option” for a description of how to create a StandAlone disk.

The following are notes on default target PCs:

- For all modes, the target PC last selected through xPC Target Explorer is the default target PC. This means that if you build a target application from a Simulink model, xPC Target builds and downloads that application for the target PC last selected. Note that a target PC is considered selected if you select any of its node components, such as **Settings**. If you delete the target PC node from xPC Target Explorer, it is still the default target PC if it is the last one selected.
- If you do not start xPC Target Explorer, or if you close and restart xPC Target Explorer, but do not select a target PC, the xpcsetup environment defines the default target PC. This means that if you build a target application from a Simulink model, xPC Target builds that application for the last target PC configured with xpcsetup.

---

**Note** The DosLoader and StandAlone options require you to purchase the xPC Target Embedded Option. Contact your MathWorks representative for further information.

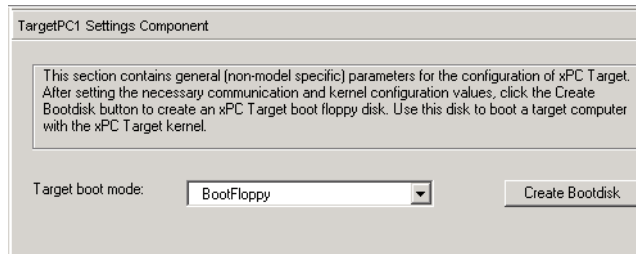
---

- 1** In the xPC Target Explorer **xPC Target Hierarchy** pane, select a target PC Configuration node. For example, select the **Configuration** node for Target PC1.

A target boot configuration pane appears in the right-most pane.

- 2** From the **Target boot mode** list, select a boot disk mode. For example, select Boot Floppy.





**3** Click the **Create BootDisk** button.

A pop-up dialog appears, requiring you to insert a formatted disk.

**4** Insert a formatted 3.5 inch disk.

xPC Target writes the xPC Target kernel and other required files to the 3.5 inch disk.

**5** When the 3.5 inch disk drive stops, remove the 3.5 inch disk.

Your next task is to install the software on the target PC and boot and test your installation. See “Installing Software, Booting a Target PC, and Testing the Target PC” on page 4-8.

## Preparing Target PCs for xPC Target System Use

- “Installing Software, Booting a Target PC, and Testing the Target PC” on page 4-8 — Describes how to install software on the target PC with the target PC boot disk and boot the target PC

### Installing Software, Booting a Target PC, and Testing the Target PC

This topic describes how to install software on a target PC, boot that PC, and test the installation and connection between the host PC and the target PC. This section assumes that you have a boot disk, created as described in “Creating a Target PC Boot Disk” on page 4-5.

- 1 Insert the new target boot disk into the corresponding target PC disk drive.
- 2 Press the reset button on the target PC.

The target PC begins to boot up.

- 3 After loading the BIOS, xPC Target boots the kernel and displays the following screen on the target PC.

<pre> Loaded App: 1MB free Memory:    124MB Mode:      loader Logging:   - StopTime:  - SampleTime: - AverageTET: - Execution: -           </pre>	<pre> ----- * xPC Target 2.6, (c)1996-2004 The MathWorks, Inc. * ----- System: Host-Target Interface is RS232 (COM1/2) System: COM2 detected, BaudRate: 115200           </pre>
---	---

- 4 In the xPC Target Explorer **xPC Target Hierarchy** pane, right-click a target, for example Target PC1.

The context menu dialog appears. This dialog lists the operations you can perform on the target PC.

- 5 Select Connect.

The Target Status pane should indicate that you are connected to the target PC.

- 6 If you want to check the connection between the host PC and target PC, in the xPC Target Explorer, from the **Tools** menu, select **Ping Target**.

Upon success, a popup dialog box appears. Click **OK** to close the box.

If the connection is established, you are ready to download a target application to the target PC. See “Downloading Target Applications to a Target PC” on page 4-11.

If the result of the target ping indicates that there is no communication between the host PC and target PC, refer to “Installation, Configuration, and Test Troubleshooting” in Chapter 13, “Troubleshooting,” of the xPC Target User’s documentation.

## Downloading and Running Target Applications

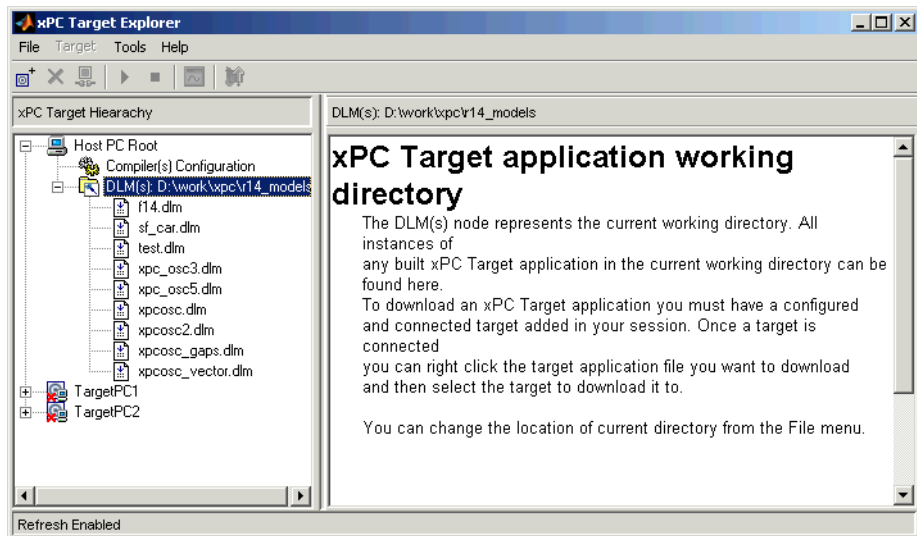
- “Changing the Prebuilt Target Application (DLM) Directory” on page 4-10 — Describes how to change the directory.
- “Downloading Target Applications to a Target PC” on page 4-11 — Describes how to download target applications to the target PC
- “Running the Target Application” on page 4-13 — Describes how to run the downloaded target application

### Changing the Prebuilt Target Application (DLM) Directory

This topic assumes that you have prebuilt target applications (DLMs) on a local disk in your host PC. It describes how to change directory to one that contains the prebuilt target applications (DLMs) you want to download to your target PCs.

- 1** In the xPC Target Explorer, left-click the **File** menu.
- 2** Select **Change current directory**.  
  
A browser is displayed.
- 3** Browse to the directory that contains the prebuilt target applications you want.
- 4** Press **OK**.

A list of the prebuilt target applications appears, as shown.



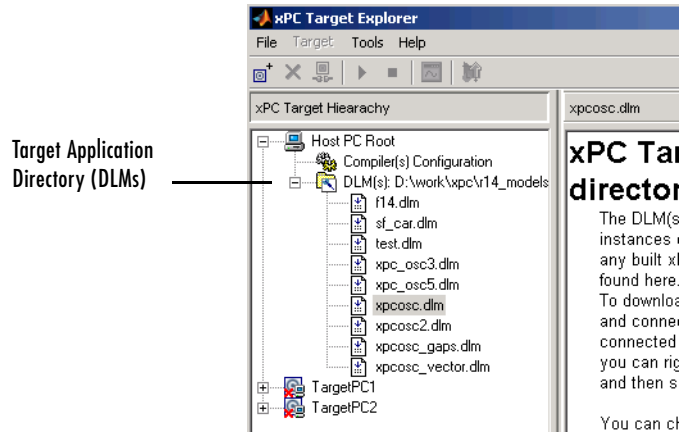
You are now ready to select and download a target application to a target PC (see “Downloading Target Applications to a Target PC” on page 4-11).

## Downloading Target Applications to a Target PC

This topic describes how to download a target application to a target PC. It assumes the following:


- In your current working directory, you have a prebuilt target application that you want to download to a target PC.
- You have installed xPC Target software and booted the target PC to which you want to download a target application.
- You have a physical connection between the xPC Target Explorer host machine and the target PC to which you want to download a target application.

- 1 In xPC Target Explorer, check that the DLM(s) node in the **xPC Target Hierarchy** has the pathname of the directory that contains the prebuilt target application you want to download to the target PC.



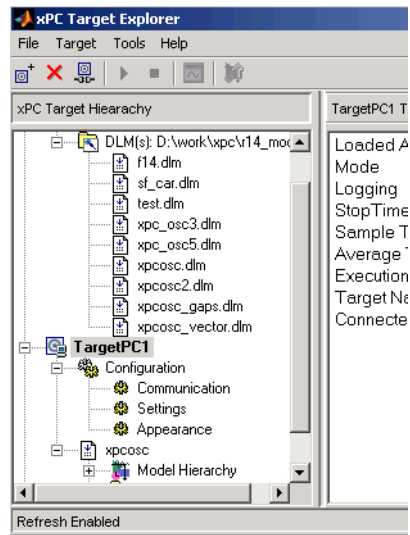
2 Right-click a target PC that you booted with xPC Target software, for example, Target PC1.

3 Select **Connect**.

The target PC icon changes and the red X is removed .

4 Left-click and drag the desired target application to the target PC to which you want to download the target application.

xPC Target Explorer downloads the target application to the target PC. A node for the target application appears in the **xPC Target Hierarchy** under the target PC node.




---

**Note** If you want to rebuild or revisit the model, click the **Go to Simulink Model** button. The Simulink model for the target application appears.

---

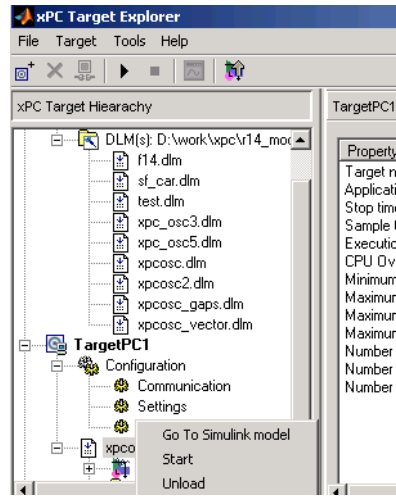
Your next task is to run the target application on the target PC. See “Running the Target Application” on page 4-13.

## Running the Target Application

This topic describes how to run a target application on a target PC. It assumes that you have already downloaded the target application to the target PC. If you have not yet done so, see “Downloading Target Applications to a Target PC” on page 4-11.

- 1 In xPC Target Explorer, right-click the downloaded target application node. For example, xpcosc.

The context menu dialog appears. This dialog lists the operations you can perform on the target application.



### 2 Select **Start**.

xPC Target Explorer starts running the loaded target application.

- 3 Stop the target application (described here) or let the target application run to the end. To stop the target application, right-click the target application node (for example, xpcosc) and select **Stop** from the list.

Now that you have successfully downloaded and run a target application, you are ready to learn about how to perform signal tracing and logging. See “Control with xPC Target Explorer” on page 4-15.



## Control with xPC Target Explorer

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 PC.

After you start running a target application, you can use signal monitoring to get signal data. This procedure uses the model `xpcosc.mdl` as an example, and assumes you created and downloaded the target application to the target PC. For meaningful values, the target application should be running.

This section describes the target application properties panel

### Manipulating Target Application Properties

- 1 In xPC Target Explorer, select the node of the loaded target application in which you are interested. For example, `xpcosc`.

The right pane changes to the target application properties pane for the application.

- 2 In this pane, you can change the following application properties
  - **Stop time**
  - **Sample time**
  - **View mode**
  - **Log mode**
- 3 Change the **Stop time** parameter to 9999.

TargetPC1 Target Application Properties

Property	Value
Target name:	TargetPC1
Application name:	xpcosc
Stop time:	Inf
Sample time:	0.00025
Execution time:	3.14625
CPU OverLoad:	none
Minimum TET:	6.70578373847443
Maximum TET:	1.42497904442581
Maximum logging ...	16666
Maximum logging ...	0
Number of signals:	7
Number of param...:	7
Number of scopes:	1

Solver

Stop time:

Sample time:

Logging

Log mode:

Time:

Output:

State:

TET:








## Menu Bar and Toolbar Contents and Shortcut Keys

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

The xPC Target Explorer menu bar has the following menus:

- **File** — General file operation options, including
  - **Add Target** — Adds a target PC to the xPC Target system
  - **Remove Target** — Removes a Target PC from the xPC Target system
  - **Change current directory** — Changes the current directory. Change directory to one that contains the prebuilt target application (DLM) you want to download to your target PCs.
  - **Close** — Close xPC Target Explorer
- **Target** — General target PC operations, including
  - **Start Application** — Starts the selected target application
  - **Stop Application** — Stops the selected target application
  - **Add a scope** — Adds a scope of type Host, Target, or File
  - **Delete scope** — Deletes a scope
  - **View host scopes** — Displays a viewer on the host PC for scopes of type host
- **Tools** — General operations, including
  - **Enable/Disable Refresh** — Enables/disables the window refresh. You should the refresh enabled to allow the xPC Target Explorer to update its display when necessary.
  - **Ping Target** — Tests the communication between the host PC and target PCs.
  - **Go to Simulink Model** — For the selected model, displays the Simulink model.
- **Help** — General product help information, including
  - **Using xPC Target** — Invokes help for xPC Target product
  - **xPC Target Explorer Help** — Invokes help for xPC Target Explorer
  - **About xPC Target** — Invokes 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

- **Add target** button 
- **Delete target** button 
- **Connect to target** button 
- **Start application** button 
- **Stop application** button 
- **Scope viewer** button 
- **Go to Simulink model** button 

The xPC Target Explorer has the following keyboard shortcuts

- Add target — **Ctrl+A**
- Remove target — **Ctrl+R**
- Close — **Ctrl+W**
- Stop/start target — **Ctrl+T**
- Ping target — **Ctrl+P**
- Delete scope — Select scope and press **Delete**

# Embedded Option

---

The xPC Target Embedded Option allows you to boot the target PC from a device other than a 3.5 inch disk drive, such as a hard disk or flash memory. It also allows you to deploy stand-alone applications on the target PC independent of the host PC. This chapter includes the following sections:

Introduction (p. 5-2)	Learn about the different types of embedded target applications you can create using the xPC Target Embedded Option
xPC Target Embedded Option Modes (p. 5-3)	Learn about the xPC Target Embedded Option modes
Embedded Option Setup (p. 5-9)	Configure xPC Target to generate embedded target applications and create a DOS system boot disk
DOSLoader Target Setup (p. 5-12)	Create a target application that boots from a device other than a 3.5 inch disk drive
Stand-Alone Target Setup (p. 5-17)	Create a target application that runs on the target PC disconnected from the host PC, and optionally, boots from a device other than a 3.5 inch disk drive

## Introduction

The xPC Target Embedded Option allows you to boot the xPC Target kernel from a 3.5 inch disk drive and other devices, including a flash disk or a hard disk drive. By using the xPC Target Embedded Option, you can configure a target PC to automatically start execution of your embedded application for continuous operation each time the system is booted. You can use this capability to deploy your own real-time applications on target PC hardware.

The xPC Target Embedded Option has two modes, DOSLoader and StandAlone, that create two different types of embedded target applications:

- DOSLoader mode allows you to boot a target PC from a device other than a 3.5 inch disk, such as a hard disk or flash memory. You can then download a target application from the host PC to the target PC.
- StandAlone mode bundles the kernel and target application into one entity that you can copy onto a 3.5 inch disk or alternate device. This allows the target PC to run as a stand-alone PC with the target application already loaded.

Additionally, the xPC Target Embedded Option allows you to deploy stand-alone GUI applications running on the host PC to control, change parameters, and acquire signal data from a target application.

Without the xPC Target Embedded Option, you can create, but not deploy, stand-alone GUI applications running on the host PC to control, change parameters, and acquire signal data from a target application. This feature uses the xPC Target API with any programming environment, or the xPC Target COM API with any programming environment, such as Visual Basic, that can use COM objects. See the xPC Target API User's and Reference documentation for further information about this feature.

## xPC Target Embedded Option Modes

The xPC Target Embedded Option extends the xPC Target base product with the two modes:

- **DOSLoader** — Use this mode of operation to start the kernel on the target PC from a 3.5 inch disk, flash disk, or a hard disk. After the target PC boots with the kernel, it waits for the host computer to download a real-time application. You can control the target application from either the host PC or the target PC. See “DOSLoader Mode Overview” on page 5-4 for further details.
- **StandAlone** — Use this mode to load the target PC with both the xPC Target kernel and a target application. Like DOSLoader mode, this mode of operation can start the kernel on the target PC from 3.5 inch disk, flash disk, or hard disk. After starting the kernel on the target PC, StandAlone mode can also automatically start the target application that you loaded with the kernel. Thus, this configuration provides complete stand-alone operation. StandAlone mode eliminates the need for a host PC and allows you to deploy real-time applications on target PCs. See “StandAlone Mode Overview” on page 5-5 for further details.

Regardless of the mode, you initially boot your target PC with DOS from any boot device, then the xPC Target kernel is started from DOS. xPC Target only needs DOS to boot the target PC and start the xPC Target kernel. DOS is no longer available on the target PC unless you reboot the target PC without starting the xPC Target kernel.

---

**Note** The xPC Target Embedded Option requires a boot device with DOS installed. It otherwise does not have any specific requirements as to the type of boot device. You can boot xPC Target from any device that has DOS installed. DOS software and license are not included with xPC Target or with the xPC Target Embedded Option.

---

Without the xPC Target Embedded Option, you can only download real-time applications to the target PC after booting the target PC from an xPC Target boot disk. You must use a target PC equipped with a 3.5 inch disk drive.

The following are some instances where you might want to use the xPC Target Embedded Option. You might have one of these situations if you deploy the target PC in a small or rugged environment:

- Target PC does not have a 3.5 inch disk drive.
- Target PC 3.5 inch disk drive must be removed after setting up the target system.

This section includes the following topics:

- “DOSLoader Mode Overview” on page 5-4
- “StandAlone Mode Overview” on page 5-5
- “Restrictions” on page 5-7

### **DOSLoader Mode Overview**

The primary purpose of DOSLoader mode is to allow you to boot from devices other than a 3.5 inch disk drive. The following summarizes the sequence of events for DOSLoader mode. For a detailed step-by-step procedure, see “DOSLoader Target Setup” on page 5-12.

- 1** Format a 3.5 inch disk.
- 2** Copy a version of DOS onto this disk and insert this DOS disk into the host PC 3.5 inch disk drive.
- 3** From the host PC MATLAB Command Window, type `xpcexplr`.
- 4** In the xPC Target Explorer **xPC Target Hierarchy** pane, select a target PC Configuration node.
- 5** From the **Target boot mode** list, select DOSLoader.
- 6** Create a boot disk. The boot disk will contain the following files:
  - DOS files — Provide your own copy of DOS to boot the target PC. For example, you can acquire DOS from FreeDOS.
  - `autoexec.bat` — xPC Target version of this file that calls the `xpcboot.com` executable to boot the xPC Target kernel.
  - `checksum.dat` — xPC Target version of this file that optimizes boot disk creation.



- \*.rtb — This file contains the xPC Target kernel. It also contains, as applicable, specifications such as serial or TCP/IP communications and the IP address of the target PC.
- xpcboot.com — Contains the xPC Target boot executable. This file executes an xPC Target application and executes the \*.rtb file.

**7** Move the boot disk to the target PC.

**8** Set up the target PC boot device such as a 3.5 inch disk, flash disk, or a hard disk drive. As necessary, transfer the contents of the boot disk to the target PC boot device.

**9** Boot the target PC.

When you boot the target PC, the target PC loads DOS, which then calls the xPC Target autoexec.bat file to start the xPC Target kernel (\*.rtb). The target PC then awaits commands from the host PC.

**10** To execute a target application, build and download one from the host PC to the target PC. DOSLoader mode does not automatically load a target application to the target PC. The xPC Target application executes entirely in protected mode using the 32-bit flat memory model.

---

**Note** This mode requires that the host PC and target PC communicate either via an RS-232 serial connection or a TCP/IP network connection.

---

## StandAlone Mode Overview

The primary purpose of the StandAlone mode is to allow you to use a target PC as a stand-alone system. The StandAlone mode enables you to deploy control systems, DSP applications, and other systems on PC hardware for use in production applications using PC hardware. Typically these production applications are found in systems where production quantities are low to moderate.

The following summarizes the sequence of events for StandAlone mode. For a detailed step-by-step procedure, see “Stand-Alone Target Setup” on page 5-17.

- 1 Format a 3.5 inch disk.
- 2 Copy a version of DOS onto this disk and insert this DOS disk into the host PC 3.5 inch disk drive.
- 3 From the host PC MATLAB window, type `xpcexplr`.
- 4 In the xPC Target Explorer **xPC Target Hierarchy** pane, select a target PC Configuration node.
- 5 From the **Target boot mode** list, select `StandAlone`.
- 6 Select and build a model.

This step creates a directory in the current working directory named `modelName_xpc_emb`.

- 7 Copy the contents of `model_name_emb` to the DOS disk. The disk should now contain the following files:
  - `DOS files` — Provide your own copy of DOS to boot the target PC. For example, you can acquire DOS from FreeDOS.
  - `*.rtb` — This file contains the xPC Target kernel. It also contains, as applicable, specifications such as serial or TCP/IP communications and the IP address of the target PC.
  - `xpcboot.com` — Contains the xPC Target boot executable. This file executes an xPC Target application and executes the `*.rtb` file.
  - `autoexec.bat` — xPC Target version of this file that calls the `xpcboot.com` executable to boot the xPC Target kernel.
- 8 Move the boot disk to the target PC.
- 9 Set up the target PC boot device such as a 3.5 inch disk, flash disk, or a hard disk drive. Transfer the contents of the boot disk to the target PC boot device.
- 10 Boot the target PC.

When you boot the target PC, the target PC loads DOS, which then calls the xPC Target `autoexec.bat` file to start the xPC Target kernel (`*.rtb`) and associated target application. If you set up the boot device to run the xPC

Target `autoexec.bat` file upon start-up, target application starts executing as soon as possible. The xPC Target application executes entirely in protected mode using the 32-bit flat memory model.

---

**Note** This mode does not require any connection between the host PC and target PC.

---

With StandAlone mode, the target PC does not communicate with the host PC. If you want to track signals on the target PC monitor, your target PC hardware configuration needs to include a monitor. To trace signals, you must add xPC Target scopes to the application before you build and transfer it to the target PC. You also need to have a target PC with a monitor. See “Adding Target Scope Blocks to Stand-Alone Applications” on page 5-18.

If you do not want to view signals on the target PC, you do not need a monitor for the target PC, nor do you need add to target scopes to the application. In this instance, your xPC Target system operates as a *black box* without a monitor, keyboard, or mouse. Stand-alone applications are automatically set to continue running for an infinite time duration or until the target computer is turned off.

## Restrictions

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

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

To satisfy these restrictions,

- Do not use additional memory managers like `emm386` or `qemmm`.
- Avoid any utilities that attempt to load in high memory (for example, `himem.sys`). If the target PC DOS environment does not use a `config.sys` file or memory manager entries in the `autoexec.bat` file, there should be no problems when running `xpcboot.com`.

- Ensure that the `xpcexplr` **TargetMouse** option setting is consistent with your hardware. Some PC hardware might use an RS-232 port for the mouse, while others use a PS2 mouse. If a mouse is not required in your application, select None as your setting for the **TargetMouse**. Choosing this setting helps prevent problems.

## Embedded Option Setup

This section includes the following topics:

- “Updating the xPC Target Environment” on page 5-9
- “Creating a DOS System Disk” on page 5-11

### Updating the xPC Target Environment

After the xPC Target Embedded Option software has been correctly installed, the xPC Target environment, visible through `xpcexplr` or `getxpcenv`, contains two additional property choices for `DOSLoader` or `StandAlone`, in addition to the default `BootDisk` that you normally use with xPC Target.

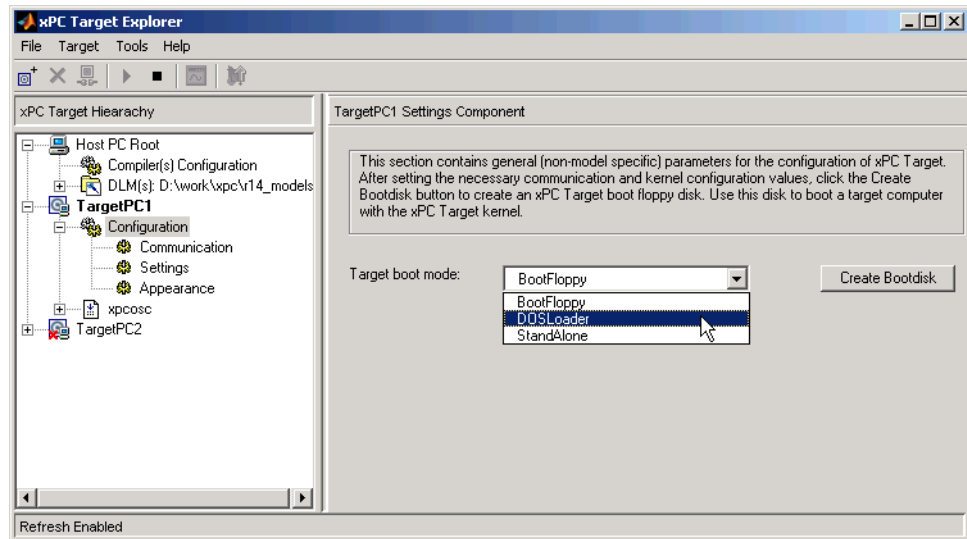
It is assumed that the xPC Target environment is already set up and working properly with the xPC Target Embedded Option enabled. If you have not already done so, confirm this now.

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

```
xpcexplr
```

In the xPC Target Explorer **xPC Target Hierarchy** pane, select a target PC Configuration node. You see the property **Target boot mode**, as well as the currently selected value. The choices are

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



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

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

The xPC Target environment is updated when you change the value. If your choice is **DOSLoader**, you must create a new target boot disk by clicking the **Create BootDisk** button. Note that this overwrites the data on the inserted target boot disk as new software modules are placed on the target boot disk. If your choice is **StandAlone**, your environment is updated, but you do not create a new target boot disk. Upon building your next real-time application, all necessary xPC Target files are saved to a subdirectory below your current working directory. This subdirectory is named with your model name with the string `'_xpc_emb'` appended, such as `xpcosc_xpc_emb`.

For more detailed information about how to use the **xPC Target Explorer** window, see Chapter 4, “xPC Target Explorer.”

## Creating a DOS System Disk

When using DOSLoader mode or StandAlone mode, you must first boot your target PC with DOS. These modes can be used from any boot device including flash disk, 3.5 inch disk drive, or a hard disk drive.

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

```
sys A:
```

---

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

---

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

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

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

## DOSLoader Target Setup

DOSLoader mode allows you to copy the xPC Target kernel to the target flash disk, remove the 3.5 inch disk drive, and then boot the xPC Target kernel. Alternatively, you can also boot the xPC Target kernel from the target PC 3.5 inch disk drive. The target application is still downloaded from the host PC. Use this mode for applications where an xPC Target host is not easily accessible.

This section includes the following topics:

- “Updating Environment Properties and Creating a Boot Disk” on page 5-12 — Select DOSLoader mode in the xPC Target Explorer window.
- “Copying the Kernel to Flash Memory” on page 5-14 — Copy the xPC Target kernel to the flash disk on the target PC and then start the kernel running.
- “Creating a Target Application for DOSLoader Mode” on page 5-16 — Create, download, and run a target application from a host PC.

### Updating Environment Properties and Creating a Boot Disk

xPC Target uses the environment properties to determine what files to create for the various target boot modes.

This procedure assumes you have serial or network communication working correctly between your host computer and a target PC. It would be helpful to successfully create a target application with the **TargetBoot** option in the **xPC Target Explorer** window set to **BootFloppy** before trying to create a kernel that boots from DOS:

- 1** On the host computer, start MATLAB.
- 2** In the MATLAB Command Window, type  

```
xpcexplr
```

The **xPC Target Explorer** window opens.
- 3** In the xPC Target Explorer **xPC Target Hierarchy** pane, select a target PC Configuration node.
- 4** From the **Target boot mode** list, select DOSLoader.



**5 Click Create BootDisk.**

A message box opens with the following message.

Insert a formatted floppy disk into your host PC disk drive and click OK to continue.

**6 Insert a 3.5 inch disk, and then click OK.**

The files checksum.dat, xpcsgo1.rtb, xpcboot.com, and autoexec.bat are copied to the disk.

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

<b>xPC Target Environment</b>	<b>HostTargetComm: RS-232</b>	<b>HostTargetComm: TCP/IP</b>
<b>TargetScope:</b> Disabled	xpcston.rtb	xpctton.rtb
<b>TargetScope:</b> Enabled	xpcsgon.rtb	xpctgon.rtb

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

Note that the autoexec.bat file should contain at least the following line:

```
xpcboot xxx.rtb
```

where xxx.rtb is the file described in the table above. View this autoexec.bat file to confirm this.

- 7** If you want to boot the target PC from the 3.5 inch disk,
  - a** Remove the 3.5 inch disk from the host PC.
  - b** Put that disk into the target PC disk drive.
  - c** Reboot the target PC. The DOS is booted from the target boot disk and the autoexec.bat files, resulting in the automatic execution of the xPC

Target loader. From this point onwards, the CPU runs in protected mode and DOS is discarded.

Otherwise, if you want to boot the target PC from flash memory instead of the 3.5 inch disk, see “Copying the Kernel to Flash Memory” on page 5-14 for a description of how to copy the kernel to flash memory. The same procedure works with flash disks and other boot devices.

---

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

---

### Copying the Kernel to Flash Memory

One method for transferring the kernel files from a host PC to a target PC is to use an external 3.5 inch disk drive.

After you create boot disk with the kernel files on a host PC, you can copy the kernel files from the 3.5 inch boot disk to the flash disk. See “Updating Environment Properties and Creating a Boot Disk” on page 5-12.

- 1 If there is a 3.5 inch disk in the external disk drive, remove it. On the target PC, press the **Reset** button.
- 2 Halt the boot process and bring up the DOS prompt. For example, if you see the message for selecting the operating system to start, select Microsoft Windows.

The boot process is stopped and a DOS prompt is displayed.

- 3 Insert the boot 3.5 inch disk with the xPC Target kernel into the target PC external 3.5 inch disk drive.
- 4 Type

```
copy A:\xpcsgo1.rtb C:\work
copy A:\xpcboot.com C:\work
copy A:\autoexec.bat C:\work
```

- 5 If you want the kernel to run when you press the **Reset** button on your target PC, save a copy of the file `C:\autoexec.bat` to a backup file, such as `C:\autoexec_back.wrk`.
- 6 Edit the file `C:\autoexec.bat` to include the following lines. Adding these commands to `C:\autoexec.bat` directs the system to execute the `autoexec.bat` file located in `C:\work`.

```
cd C:\work
autoexec
```

---

**Note** Do not confuse `C:\work\autoexec.bat` with `C:\autoexec.bat`. The file `C:\work\autoexec.bat` includes the command `xpcboot.com` to start the xPC Target kernel. The file `C:\autoexec.bat` includes the files you want the system to execute when the system starts up.

---

- 7 Edit the file `C:\autoexec.bat` to include the lines

```
cd C:\work
autoexec
```

- 8 Remove the 3.5 inch disk, and then, on the target PC, press the **Reset** button.

The sequence of calls during the boot process is

- a `C:\autoexec.bat`
- b `C:\work\autoexec.bat`
- c `C:\work\xpcboot.com`
- d `C:\work\xpmsg01.rtb`

### Creating a Target Application for DOSLoader Mode

For DOSLoader mode, a target application is created on a host PC and downloaded to your target PC.

After you set the Simulink and Real-Time Workshop<sup>®</sup> parameters for code generation with xPC Target in your Simulink model, you can use xPC Target with DOSLoader mode to create a target application:

- 1** In the MATLAB window, type the name of a Simulink model. For example, type  
`xpc_osc3`  
  
A Simulink window opens with the model.
- 2** From the **Tools** menu, point to **Real-Time Workshop**, and then click **Build Model**.
- 3** Real-Time Workshop and xPC Target create a target application and download it to your target PC.

## Stand-Alone Target Setup

StandAlone mode combines the target application with the kernel and boots them together on the target PC from flash memory (or alternatively, the target PC 3.5 inch disk drive). The host PC does not need to be connected to the target PC. This section includes the following topics:

- “Updating Environment Properties” on page 5-17 — Select StandAlone mode in the xPC Target Explorer window.
- “Adding Target Scope Blocks to Stand-Alone Applications” on page 5-18 — Add target scope blocks to models to monitor signal data.
- “Creating a Kernel/Target Application” on page 5-21 — On the host PC, create a stand-alone application.
- “Copying the Kernel/Target Application to Flash Disk” on page 5-22 — Copy the combined xPC Target kernel and target application to the flash disk on the target PC.

### Updating Environment Properties

xPC Target uses the environment properties to determine what files to create for the various target boot modes.

This procedure assumes you have serial or network communication working correctly between your host computer and a target PC. It would be helpful to successfully create a target application with the **TargetBoot** option in the **xPC Target Explorer** window set to **BootFloppy** before trying to create a stand-alone application:

- 1 On the host computer, start MATLAB.
- 2 In the MATLAB window, type  

```
xpcexplr
```

The **xPC Target Explorer** window opens.
- 3 In the xPC Target Explorer **xPC Target Hierarchy** pane, select a target PC Configuration node.
- 4 From the **Target boot mode** list, select **DOSLoader**.

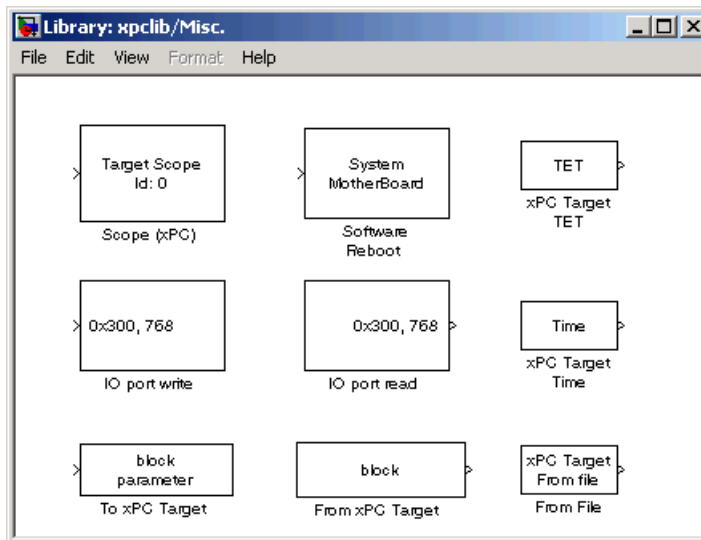
- From the **Target boot mode** list, choose StandAlone.

xPC Target updates the environment properties, and the build process is ready to create a stand-alone kernel/target application.

For StandAlone mode, you do not create an xPC Target boot disk. Instead, you copy files created from the build process onto a formatted 3.5 inch disk.

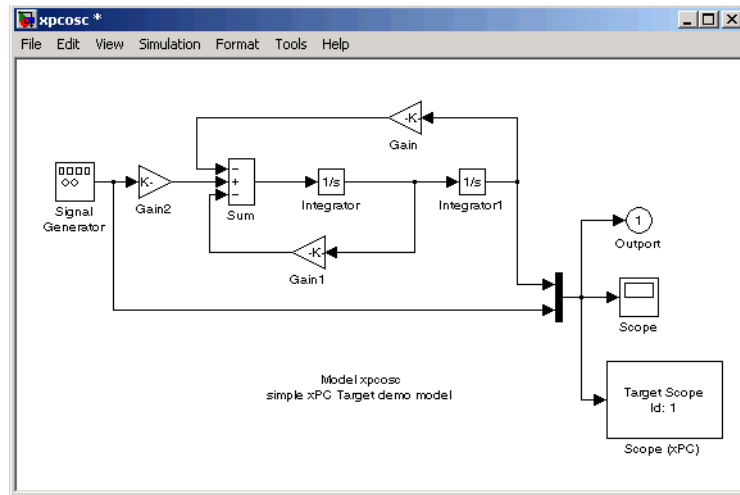
## Adding Target Scope Blocks to Stand-Alone Applications

When using xPC Target Embedded Option with StandAlone mode, you can optionally use scopes of type target or file to trace signals and display them on the target screen. Because host-to-target communication is not supported with StandAlone mode, scope objects of type target or file must be defined within the Simulink model before the xPC Target application is built. xPC Target offers the **Scope (xPC)** block for such purposes.

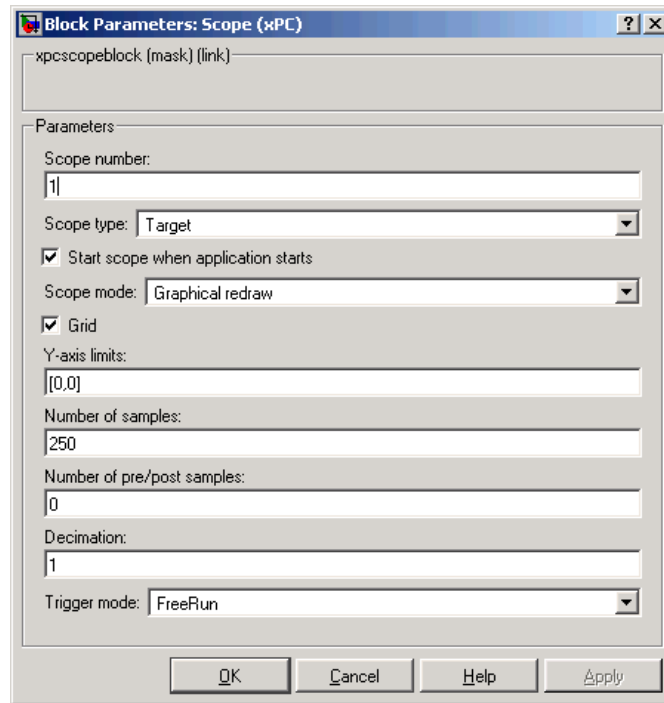


To add a **Scope (xPC)** block to a Simulink model,

- Copy the **Scope (xPC)** block into your block diagram and connect the signals you would like to view to this block. You can use multiple signals as long as you use a Mux block to bundle them.



- 2** Edit the **Scope (xPC)** dialog box and confirm that the check box entry for **Start scope when application starts** is selected, as shown in the following dialog box.



This setting is required to enable target scopes to begin operating as soon as the application starts running. This setting is important because the host PC is not available in StandAlone mode to issue a command to start scopes.

- 3 Ensure that the **Scope type** field is Target or File.
- 4 Save the model.

Your next task is to create a kernel/target application. See “Creating a Kernel/Target Application” on page 5-21.



## Creating a Kernel/Target Application

Use xPC Target with StandAlone mode to create a combined kernel and target application with utility files. A combined kernel and target application allows you to disconnect your target PC from a host PC and run stand-alone applications.

After you set the Simulink and Real-Time Workshop parameters for code generation with xPC Target in your Simulink model, you can use xPC Target with StandAlone mode to create a target application:

- 1 In the MATLAB window, type the name of a Simulink model. For example, type

```
xpc_osc3
```

A Simulink window opens with the model.

- 2 From the **Tools** menu, point to **Real-Time Workshop**, and then click **Build Model**.

Real-Time Workshop and xPC Target create a directory `xpc_osc3_xpc_emb` with the following files:

- `autoexec.bat` — This file is automatically invoked by DOS. It then runs `xpcboot.com` and the `*.rtb` file.
- `xpc_osc3.rtb` — This image contains the xPC Target kernel and your target application.
- `xpcboot.com` — This file is a static file that is part of xPC Target Embedded Option.

- 3 Copy the preceding files to a formatted 3.5 inch disk.
- 4 If you want to boot the target PC from the 3.5 inch disk,
  - a Remove the 3.5 inch disk from the host PC.
  - b Put that disk into the target PC disk drive.
  - c Reboot the target PC. The DOS is booted from the target boot disk and the `autoexec.bat` files, resulting in the automatic execution of the xPC

Target loader. From this point onwards, the CPU runs in protected mode and DOS is discarded.

Otherwise, if you want to boot the target PC from flash memory instead of the 3.5 inch disk, see “Copying the Kernel to Flash Memory” on page 5-14 for a description of how to copy the kernel to flash memory. The same procedure works with flash disks and other boot devices.

### Copying the Kernel/Target Application to Flash Disk

You build a target application on a host PC using Real-Time Workshop, xPC Target, and a C/C++ compiler. One method for transferring the files from the host PC to a target PC is to use an external 3.5 disk drive.

After you build a stand-alone application on a host PC, you can copy files from a 3.5 inch disk to the flash disk. If you have not already copied the necessary files to a 3.5 inch disk, see “Creating a Kernel/Target Application” on page 5-21.

- 1 If there is a 3.5 disk in the target PC external disk drive, remove it. On the target PC, press the **Reset** button.
- 2 Halt the boot process and bring up the DOS prompt. For example, if you see the message for selecting the operating system to start, select Microsoft Windows.

The boot process is stopped and a DOS prompt is displayed.

- 3 Insert the 3.5 inch disk with the stand-alone application and utility files into the external 3.5 inch disk drive of the target PC.

#### 4 Type

```
copy A:\xpc_osc3.rtb C:\work
copy A:\xpcboot.com C:\work
copy A:\autoexec.bat C:\work
```

- 5 If you want your stand-alone application to run when you press the **Reset** button on your target PC, save a copy of the file C:\autoexec.bat to a backup file, such as C:\autoexec\_back.wrk.

- 
- 6** Edit the file `C:\autoexec.bat` to include the following lines. Adding these commands to `C:\autoexec.bat` directs the system to execute the `autoexec.bat` file located in `C:\work`.

```
cd C:\work
autoexec
```

---

**Note** Do not confuse `C:\work\autoexec.bat` with `C:\autoexec.bat`. The file `C:\work\autoexec.bat` includes the command `xpcboot.com` to start the xPC Target kernel and stand-alone application. The file `C:\autoexec.bat` includes the files you want the system to execute when the system starts up.

---

- 7** Remove the 3.5 inch disk, and then, on the target PC, press the **Reset** button.

The sequence of calls during the boot process is

- a** `C:\autoexec.bat`
- b** `C:\work\autoexec.bat`
- c** `C:\work\xpcboot.com`
- d** `C:\work\<application>.rtb`

- 8** On the target PC keyboard, press the spacebar.

A command line opens on the target PC screen.

For a complete list of target PC commands, see “Using the Target PC Command-Line Interface” on page 7-1.



# Software Environment and Demos

---

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 GUI environment.

xPC Target provides a number of demos that help you understand the product.

Using Environment Properties and Functions (p. 6-2)

Common tasks within the xPC Target software environment

xPC Target Demos (p. 6-6)

List of xPC Target demos, accessible from the MATLAB Command Window

## Using Environment Properties and Functions

Use the **xPC Target Explorer** window or the MATLAB Command Window to enter environment properties that are independent of your model. This section includes the following topics:

- “Getting a List of Environment Properties” on page 6-2
- “Changing Environment Properties with a Graphical Interface” on page 6-3
- “Changing Environment Properties with a Command-Line Interface” on page 6-5

Refer to the function `getxpcenv` for a reference of the environment properties and functions.

To enter properties specific to your model and its build procedure, see “Entering the Real-Time Workshop Parameters” on page 3-40. 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. Use the following procedure to get a list of the property names, their allowed values, and their current values:

- 1 In the MATLAB window, type

```
setxpcenv
```

MATLAB displays a list of xPC Target environment properties and the allowed values. For a list of the properties, see the function `getxpcenv` on page 16-54.

- 2 Type

```
getxpcenv
```

MATLAB 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. In the MATLAB window, type `xpcexplr`.

## Changing Environment Properties with a Graphical Interface

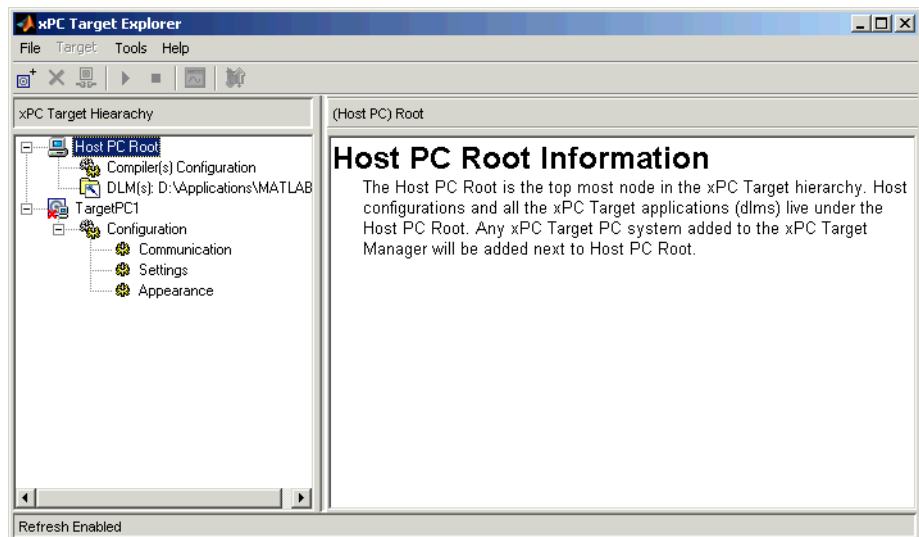
xPC Target lets you define and change environment properties. These properties include the path to the C/C++ compiler, the host PC COM port, the logging buffer size, and many others. Collectively these properties are known as the xPC Target environment.

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

- 1 In the MATLAB window, type

```
xpcexplr
```

MATLAB 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 equivalent nodes to the **xPC Target Hierarchy** pane. The foremost node is the HostPC node. It represents the host PC. The next node of importance is the TargetPC node. Each time you add a target PC 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 PC node.

The right pane displays information that reflects an 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 property **Target boot mode**. If your license does not include the xPC Target Embedded Option, the **Target boot mode** box is grayed out, with BootFloppy as your only selection. With the xPC Target Embedded Option, you have the additional choices of DOSLoader and StandAlone.

- 2** Change properties in the environment 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.



## Changing Environment Properties with a Command-Line Interface

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

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

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

- 1 In the MATLAB window, type

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

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

HostTargetComm	:RS232	up to date
RS232HostPort	:COM1	COM2
RS232Baudrate	:115200	up to date

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

- 2 In the MATLAB window, type

```
updatexpcenv
```

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

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

## xPC Target Demos

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

Demo	Filename
Parameter Sweep	parsweepdemo
Signal tracing using free-run mode	scfreerundemo
Signal tracing using software triggering	scsoftwaredemo
Signal tracing using signal triggering	scsignaldemo
Signal tracing using scope triggering	scscopedemo
Signal tracing using the target scope	tgscopedemo
Pre/post triggering of xPC Target scopes	scprepostdemo
Time- and value-equidistant data logging	dataloggingdemo

### To Locate or Edit a Demo Script

- 1 In the MATLAB Command Window, type

```
which scfreerundemo
```

MATLAB displays the location of the M-file.

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

- 2 Type

```
edit scfreerundemo
```

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

# Using the Target PC Command-Line Interface

---

You can interact with the xPC Target environment through the target PC command window. xPC Target provides a limited set of commands that you can use to work with the target application after it has been loaded to the target PC, and to interface with the scopes for that application.

Target PC Command-Line Interface  
(p. 7-2)

Enter commands on the target PC for stand-alone applications that are not connected to the host PC

## Target PC Command-Line Interface

This interface is useful with stand-alone applications that are not connected to the host PC. You can type commands directly from a keyboard on the target PC. As you start to type at the keyboard, a command window appears on the target PC screen. This section includes the following topics:

- “Using Target Application Methods on the Target PC” on page 7-2
- “Manipulating Target Object Properties from the Target PC” on page 7-3
- “Manipulating Scope Objects from the Target PC” on page 7-4
- “Manipulating Scope Object Properties from the Target PC” on page 7-6
- “Aliasing with Variable Commands on the Target PC” on page 7-6

For a complete list of target PC commands, refer to Chapter 14, “Target PC Commands.”

### Using Target Application Methods on the Target PC

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

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

- 1** On the target PC, press **C**.

The target PC 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 PC 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 PC

xPC Target 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 PC.

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

The target PC activates the command window.

- 2 Type a target command. For example, to change the frequency of the signal generator (parameter 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 PC returns the current properties of the target object.

---

**Note** The target PC 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 PC

xPC Target 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 PC.

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

The target PC activates the command window.

- 2** Type a scope command. For example, to add a target scope (scope 2) in the model `xpcosc`, type

```
addscope 2
```

xPC Target adds another scope monitor to the target PC 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 as many signals as necessary 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 PC, 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 PC

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 PC.

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

The target PC activates the command window.

- 2 Type a 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 PC

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 PC, you can create target PC variables.

- 1 On the target PC 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 PC 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.



# Working with Target PC Files and File Systems

---

xPC Target scopes of type `file` create files on the target PC. To work with these files from the host PC, you need to work with the `xpctarget.ftp` and `xpctarget.fs` objects. The `xpctarget.ftp` object allows you to perform basic FTP-like operations on the target PC file system. The `xpctarget.fs` object allows you to perform file system-like operations on the target PC file system. This chapter contains the following topics:

Introduction (p. 8-2)

Introduction to the `xpctarget.ftp` and `xpctarget.fs` objects

FTP and File System Objects (p. 8-4)

Description of FTP and file system objects

Using `xpctarget.ftp` Objects (p. 8-5)

Use the MATLAB Command Window to use file object methods to access the target PC files from the host PC

Using `xpctarget.fs` Objects (p. 8-8)

Use the MATLAB Command Window to use file system methods to access the target PC file system from the host PC

## Introduction

The xPC Target scope object of type `file` always writes acquired signal data to a file on the target PC. You cannot direct the scope to write the data to a file on the xPC Target host PC. Once xPC Target has written the signal data file to the target PC, you can access the contents of the file for plotting or other inspection from the host PC. xPC Target can write data files to

- The C:\ or D:\ drive of the target PC. This must be an Integrated Device Electronics (IDE) drive. xPC Target supports file systems of type FAT-12, FAT-16, or FAT-32.
- A 3.5 inch disk drive.

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 PC system file, in one of the following ways:

- If you are running the target PC as a stand-alone system, you can access that file by rebooting the target PC under an operating system such as DOS and accessing the file through the operating system utilities.
- If you are running the target PC in conjunction with a host PC, you can access the target PC file from the host PC by representing that file as an `xpctarget.ftp` object. Through the MATLAB interface, use `xpctarget.ftp` methods on that file object. The `xpctarget.ftp` object methods are file transfer operations such as `get` and `put`.
- If you are running the target PC in conjunction with a host PC, you can access the target PC file from the host PC by representing the target PC 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 PC file system information for the disk and disk buffers.

This chapter describes procedures on how to use the `xpctarget.ftp` and `xpctarget.fs` methods for common operations. See Chapter 16, “Function Reference,” for a reference of the methods for these objects.

---

**Note** This section focuses primarily on working with the target PC data files that you generate from an xPC Target scope object of type `file`.

---

## FTP and File System Objects

xPC Target uses two objects, `xpctarget.ftp` (FTP) and `xpctarget.fs` (file system), to work with files on a target PC. You use the `xpctarget.ftp` object to perform standard file transfer operations between the host and target PC. You use the `xpctarget.fs` object to access the target PC 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 scope of type `file`.

To create an `xpctarget.ftp` object,

- Ensure that a target application exists on the target PC. xPC Target requires that an application exist on the target PC before you can access the target PC file.
- Use the file object constructor function `xpctarget.ftp`. In the MATLAB Command Window, type `f = xpctarget.ftp`.

xPC Target uses a file system object on the host PC to represent the target PC file system. You use file system objects to work with that file system from the host PC.

To create an `xpctarget.fs` object,

- Ensure that a target application exists on the target PC. xPC Target requires that an application exist on the target PC before you can access the target PC file.
- Use the file 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`. xPC Target creates the `xpctarget.fs` base object when you create either an `xpctarget.ftp` or `xpctarget.fs` object.

## Using xpctarget.ftp Objects

The `xpctarget.ftp` object enables you to work with any file on the target PC, including the data file that you generate from an xPC Target scope object of type `file`. The `xpctarget.ftp` object has methods that allow you to use

- `cd` to change directories
- `dir` to list the contents of the current directory
- `get (ftp)` to retrieve a file from the target PC to the host PC
- `mkdir` to make a directory
- `put` to place a file from the host PC to the target PC
- `pwd` to get the current working directory path
- `rmdir` to remove a directory

The procedures in this section assume that the target PC has a signal data file created by an xPC Target scope of type `file`. This file has the pathname `C:\data.dat`. See “Simulink Model” in Chapter 3 of the Getting Started with xPC Target documentation and “Signal Tracing with xPC Target Scope Blocks” in Chapter 3 of this document for additional details.

This section includes the following topics:

- “Listing the Contents of the Target PC Directory” on page 8-5
- “Retrieving a File from the Target PC to the Host PC” on page 8-6
- “Copying a File from the Host PC to the Target PC” on page 8-7

xPC Target also provides methods that allow you to perform file system-type manipulations, such as opening and reading files. For a complete list of these methods, see “Using xpctarget.fs Objects” on page 8-8.

### Listing the Contents of the Target PC Directory

You can list the contents of the target PC directory by using xPC Target methods on the host PC 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 directory. To get the results in an M-by-1 structure, use a syntax like `y=dir(f)`. See the `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 directory. You get a result like the following:

```
ans =  
C:\
```

- 3 Type the following to list the contents of this directory:

```
dir(f)
```

### Retrieving a File from the Target PC to the Host PC

You can retrieve a copy of a data file from the target PC by using xPC Target methods on the host PC 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 PC 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 PC to the Target PC**

You can place a copy of a file from the host PC by using xPC Target methods on the host PC 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 PC to the target PC 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 retrieve the file and save that file to the variable `data`.

```
f.put('data2.dat');
```

## Using `xpctarget.fs` Objects

The `fs` object enables you to work with the target PC file system. The `fs` object has methods that allow you to use

- `cd` to change directories
- `dir` to list the contents of the current directory
- `diskinfo` to get information about the current disk
- `fclose` to close a file (similar to MATLAB `fclose`)
- `fileinfo` to get information about a particular file
- `filetable` to get information about files in the file system
- `fopen` to open a file (similar to MATLAB `fopen`)
- `fread` to read a file (similar to MATLAB `fread`)
- `fwrite` to write a file (similar to MATLAB `fwrite`)
- `getfilesize` to get the size of a file in bytes
- `mkdir` to make a directory
- `pwd` to get the current working directory path
- `removefile` to remove a file from the target PC
- `rmdir` to remove a directory

Useful global utility:

- `readxpcfile`, to interpret the raw data from the `fread` method

The procedures in this section assume that the target PC has a signal data file created by an xPC Target scope of type `file`. This file has the pathname `C:\data.dat`.

This section includes the following topics:

- “Retrieving the Contents of a File from the Target PC to the Host PC” on page 8-9
- “Removing a File from the Target PC” on page 8-11
- “Getting a List of Open Files on the Target PC” on page 8-11
- “Getting Information about a File on the Target PC” on page 8-12
- “Getting Information about a Disk on the Target PC” on page 8-13

xPC Target also provides methods that allow you to perform file transfer operations, such as putting files on and getting files from a target PC. For a description of these methods, see “Using xpctarget.fs Objects” on page 8-5.

## Retrieving the Contents of a File from the Target PC to the Host PC

You can retrieve the contents of a data file from the target PC by using xPC Target methods on the host PC 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 retrieve the contents of a file named `data.dat` from the target PC 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 Bytes” on page 8-10.

**Converting xPC Target File Format Content to Bytes**

If you have xPC Target file format content that you want to view or plot, you need to convert that content to bytes. xPC Target provides the script `readxpcfile.m` to convert xPC Target format content.

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 PC to the Host PC” on page 8-9):

- 1** In the MATLAB window, change directory to the directory 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 value of the original Simulink signal at the specified time during target execution.

You can view or examine the signal data. You can also plot the data with `plot(new_data2.data)`.

## Removing a File from the Target PC

You can remove a file from the target PC by using xPC Target methods on the host PC 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 PC 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 PC.

```
fsys.removefile('data2.dat');
```

or

```
removefile(fsys,'data2.dat');
```

## Getting a List of Open Files on the Target PC

You can get a list of open files on the target PC file system from the host PC by using xPC Target methods on the host PC for the `xpctarget.fs` object. Do this to ensure you do not have files opened unnecessarily. The target PC 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.DAT
```

- 3 The table returns the open file handles in hexadecimal. To convert a handle to one that other `xpctarget.fs` methods, such as `fclose`, 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 PC

You can display information for a file on the target PC file system from the host PC by using xPC Target methods on the host PC 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 `h`.

**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 PC**

You can display information for a disk on the target PC file system from the host PC by using xPC Target methods on the host PC 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
```



# Graphical User Interfaces

---

You can run and test your target application using the MATLAB command-line interface or the Simulink block diagram for your application. You cannot modify these interfaces, but you can use special blocks provided with xPC Target to interface signals and parameters from a target application to create a custom graphical user interface (GUI) for your xPC Target application. This chapter includes the following sections:

xPC Target Interface Blocks to  
Simulink Models (p. 9-2)

Overview describing the software products you can use  
with the To xPC Target and From xPC Target blocks

Interface with Dials & Gauges Blockset  
(p. 9-8)

Example using Simulink, the Dials & Gauges Blockset,  
and the To and From xPC Target blocks to create a  
custom user interface to a target application

## xPC Target Interface Blocks to Simulink Models

You can use Simulink to create a custom graphical user interface (GUI) for your xPC Target application. You do this by creating an user interface model with Simulink and add-on products like the Dials & Gauges Blockset, Virtual Reality Toolbox, and Altia Design (a third-party product). This section includes the following topics:

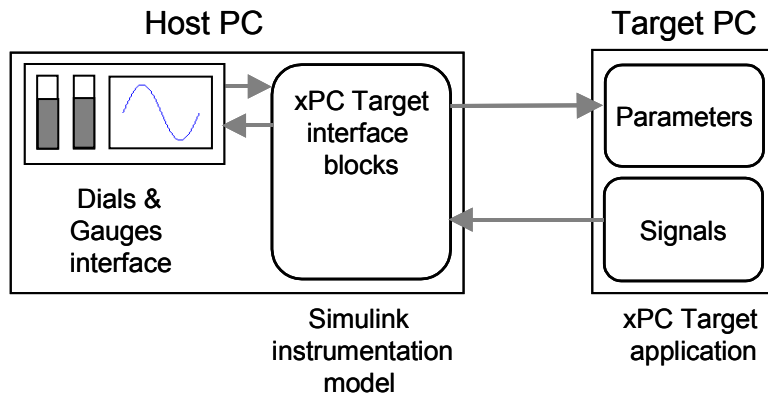
- “Simulink User Interface Model” on page 9-2 — Simulink model with xPC Target interface blocks to your target application and interface blocks to graphical elements and interfaces
- “Creating a Custom Graphical Interface” on page 9-4 — The process for creating a custom graphical interface includes tagging parameters and signals, and then creating a Simulink user interface model with interface blocks to these parameters and signals
- “To xPC Target Block” on page 9-5 — Simulink blocks that take new parameter values from graphical elements and download those values to your target application
- “From xPC Target Block” in Chapter 9 — Simulink blocks that upload signal data from your target application and pass that data to graphical elements for visualization

### Simulink User Interface Model

A user interface model is a Simulink model containing Simulink blocks from add-on products and interface blocks from xPC Target. This user interface model can use Dials & Gauges or it can connect to a custom graphical interface (using Virtual Reality Toolbox or Altia products). The user interface model runs on the host PC and communicates with your target application running on the target PC using To xPC Target and From xPC Target blocks.

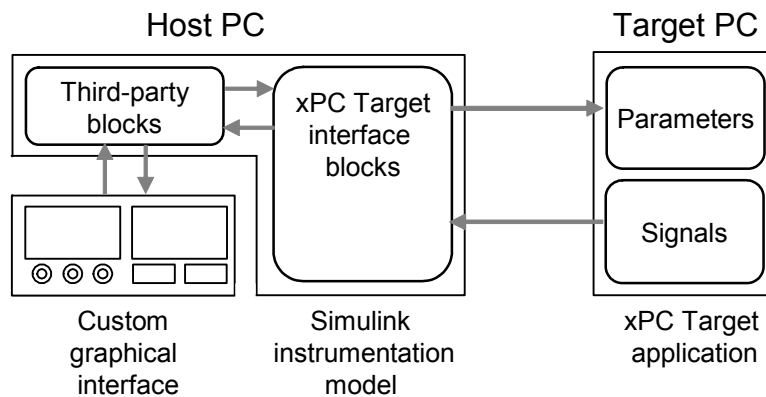
The graphical user interface allows you to change parameters by downloading them to the target PC, and to visualize signals by uploading data to the host PC.

**Dials & Gauges Blockset** — The Dials & Gauges Blockset enables you to include graphic controls and displays directly in your Simulink user interface model. This user interface model is the graphical user interface to your target application.



**Virtual Reality Toolbox** — The Virtual Reality Toolbox 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 with a graphical user interface. This graphical user 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.



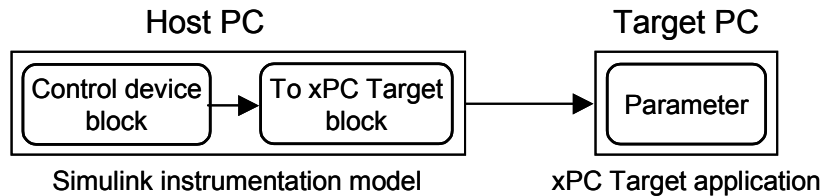
## Creating a Custom Graphical Interface

xPC Target provides Simulink interface blocks to connect graphical interface elements to your target application. The steps for creating your own custom graphical interface are listed below. For more information, see “Interface with Dials & Gauges Blockset” on page 9-8.

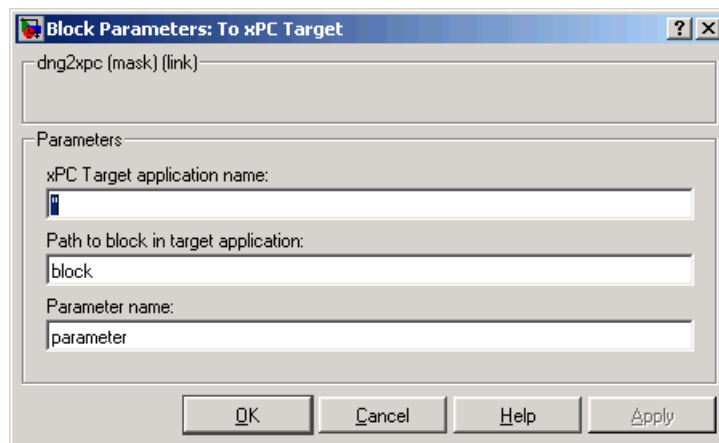
- 1 In the Simulink target application model, decide which block parameters and block signals you want to have access to through graphical interface control devices and graphical interface display devices.
- 2 Tag all block parameters in the Simulink model that you want to be connected to a control device. See “Marking Block Parameters” on page 9-13.
- 3 Tag all signals in the Simulink model that you want to be connected to a display device. See “Marking Block Signals” on page 9-15.
- 4 In MATLAB, run the function `xpcsliface('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. See “Creating a User Interface Model” on page 9-19.
- 5 To the user interface template model, add Simulink interface blocks from add-on and third-party products (Dials & Gauges Blockset, Virtual Reality Toolbox, Altia Design). Connect these blocks to the xPC Target interface blocks (To PC Target and From xPC Target). The To xPC Target blocks on the left should be connected to control devices, and the 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. See “Running a Target Application with a User Interface Model” on page 9-23.

### 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 while it is running.



This block is implemented as an M-file 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 command-line interface. This block is available from the **xpcLib/Misc** block sublibrary.



**Note** The use of To xPC Target blocks require a connection between the host and target PC. If there is no connection between the host and target PC, operations such as opening a model that contains these blocks or copying these blocks within or between models, will take significantly longer than normal.

## Block Parameters

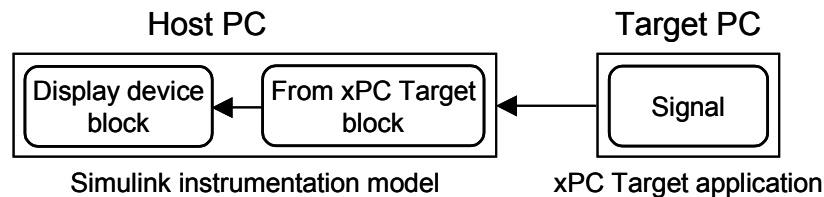
**xPC Target application name** — The function `xpcsliface` automatically enters a name entry for this parameter. It is the same name as the Simulink model that xPC Target uses to build the target application.

**Path to block in model running on xPC target** — The function `xpcsliface` automatically enters an entry for this parameter and uses it to access the block identifier.

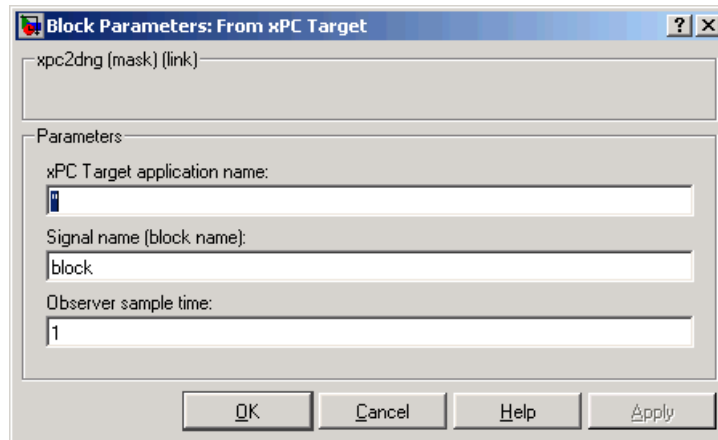
**Parameter name** — The function `xpcsliface` automatically determines the entry for this parameter and enters it. Note that the parameter name might not match the label name for that parameter in the **Block Parameters** dialog box. For example, the label name for a gain block is `Constant value`, but the parameter name is `Value`.

## 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 signal tracing capability of the xPC Target command-line interface and is implemented as an M-file S-function. This block is available from the `xpc/lib/Misc` sublibrary.



---

**Note** The use of From xPC Target blocks require a connection between the host and target PC. If there is no connection between the host and target PC, operations such as opening a model that contains these blocks or copying these blocks within or between models, will take significantly longer than normal.

---

### Block Parameters

**xPC Target application name** — The function `xpcsliface` automatically enters a name entry for this parameter. It is the same name as the Simulink model that xPC Target uses to build the target application.

**Signal name (block name)** — The function `xpcsliface` automatically enters a name entry for this parameter.

**Observer sample time** — The function `xpcsliface` automatically enters the sample time for the Simulink block with this signal. It can be equal to the model base sample time or a multiple of the base sample time.

## Interface with Dials & Gauges Blockset

Use the Dials & Gauges Blockset and xPC Target interface blocks to create a custom graphical user interface (GUI) for your xPC Target application. This method allows you to create a user interface (UI) within a Simulink model.

This section includes the following topics:

- “Introduction to the Dials & Gauges Blockset” on page 9-8 — What is the Dials & Gauges Blockset?
- “Target Application Model Description” on page 9-11 — An example control system using a water tank, pump, drain, and valve controller
- “Creating a Target Application Model” on page 9-12 — A Simulink model for your physical system and controller. xPC Target uses this model to create a target application.
- “Marking Block Parameters” on page 9-13 — Parameters you want to change using dial blocks
- “Marking Block Signals” on page 9-15 — Signals you want to visualize using gauge blocks
- “Creating a User Interface Model” on page 9-19 — Create a Simulink user interface model as a graphical user interface to a target application.
- “Adding Dials & Gauges Blockset” in Chapter 9 — Dials for changing parameters and gauges for visualizing signals
- “Creating a Target Application” on page 9-21 — Create a real-time application from your Simulink application model using Real-Time Workshop and a C compiler.
- “Running a Target Application with a User Interface Model” on page 9-23 — Start the target application running in real time and the user interface model running in nonreal time.

### Introduction to the Dials & Gauges Blockset

Using the Dials & Gauges Blockset as a graphical interface to your target application simplifies and enhances user interaction by providing an alternative to using a command-line interface.

The Dial & Gauges Blockset consists of a library of blocks that you add to your Simulink model. When used with xPC Target, components from the Dials &



Gauges Blockset provide a graphical user interface that you use from your host PC. The graphical user interface allows you to change parameters that are then downloaded to your target PC. Other components from the Dials & Gauges Blockset provide animation of data uploaded from the target PC. While you change parameters or view the animation, your target application continues to run in real time.

The Dials & Gauges Blockset includes control blocks and display blocks.

**Control blocks** — A control block is a block that you use to provide an input using a mouse or keyboard. For example, a control block could be a slider or a dial. Its output drives other blocks in the target application model and provides a visual display of the output parameters.

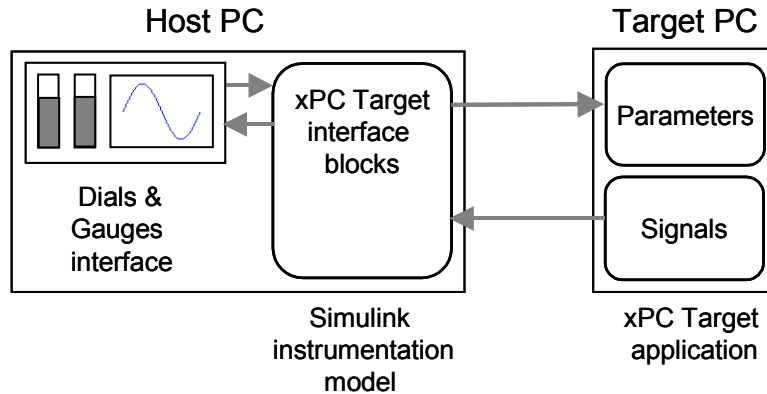
**Display blocks** — A display block is a block that you use to view signals on your host computer. For example, a display block could be a tachometer or a gauge. A display block receives its input from the target application model and provides a visual display of the input signal.

### Dials & Gauges Blockset Blocks

The Dials & Gauges Blockset consists of approximately sixty blocks grouped into the following block collections:

- **Angular Gauges** — Controls that show their input value graphically along an arc of a circle.
- **Buttons & Switches** (toggle)— Two-state controls that change state when you click on them
- **Knobs & Selectors** — Controls that change their output values when you drag a knob around a circle
- **LEDs** — Controls that use graphical elements to imitate light-emitting diodes (LEDs)
- **Linear Gauges** —Displays that graphically show an input value along a linear scale
- **Numeric Displays** — Controls that display the numeric value of their input signals
- **Percent Indicators** — Controls that display percentages and ratios.
- **Sliders** — Controls that model a knob sliding along a bar and that output the numerical value corresponding to the knob position.

- **Strip Chart** — A display where the input value traces from left to right. This environment allows you to interact with your target application using a graphical user interface while the target application is running in real time on the target PC.



The procedure for using xPC Target with the Dial & Gauges Blockset consists of the following steps:

- 1 Install software for MATLAB, Simulink, Real-Time Workshop, xPC Target, third-party C compiler, and Dials & Gauges Blockset.
- 2 Create a Simulink target application model containing model equations describing the dynamic behavior of the application you want to run in real time on the target PC. See “Creating a Target Application Model” on page 9-12.
- 3 In your target application model, tag block properties and block signals. See “Marking Block Parameters” on page 9-13 and “Marking Block Signals” on page 9-15.
- 4 Create a Simulink user interface model using Dials & Gauges Blockset blocks as graphical user interface components (for example, control and display devices). See “Creating a User Interface Model” on page 9-19 and “Adding Dials & Gauges Blockset” on page 9-20.

- 5 Create a target application and download it to the target PC. See “Creating a Target Application” on page 9-21.
- 6 Start a simulation of the user interface model in nonreal time, and then start the target application running in real time. See “Running a Target Application with a User Interface Model” on page 9-23.

## Target Application Model Description

xPC Target includes the Simulink model `xpctank.mdl`. This is a model of a water tank with a pump, drain, and valve controller. See “Creating a Target Application Model” on page 9-12 for an implementation of this model.

**TankLevel** — The water level in the tank is modeled using a limited integrator named TankLevel.

**PumpSwitch** — The pump can be turned off manually to override the action of the controller. This is done by setting `PumpSwitch = 0`. When `PumpSwitch = 1`, the controller is able to use the ControlValve to pump water into the tank.

**ValveSwitch (drain valve)** — The tank has a drain valve that allows water to flow out of the tank. Think of this as water usage or consumption that reduces the water level. This behavior is modeled with the constant block named ValveSwitch, the gain block Gain2, and a summing junction. The minus sign on the summing junction has the effect of producing a negative flow rate (drain) that reduces the water level in the tank.

Although the ValveSwitch is modeled as a constant block, you can later alter its value using the Dials & Gauges Blockset. When `ValveSwitch = 0` (closed), the valve is closed and water cannot flow out of the tank. When `ValveSwitch = 1` (open), the valve is open and the water level is reduced by draining the tank.

**Controller** — The controller is very simple. It is a bang-bang controller and can only maintain the selected water level by turning the control valve (pump valve) on or off. A water level set point is used to define the desired median water level. Hysteresis is provided to enable the pump to avoid high-frequency on and off cycling. This is done using symmetric upper and lower bounds that are offsets to the median set point. As a result, the controller turns the control valve (pump valve) on whenever the water level is below the set point minus the offset. The summing junction compares this lower bound against the tank water level to determine whether or not to open the control valve. If the pump is turned on (`PumpSwitch = 1`), water is pumped into the tank. When the water

level reaches or exceeds the set point plus the upper bound, the controller turns off the control valve. Regardless of whether the pump is on or off, water stops pumping into the tank.

**Scope blocks** — A standard Simulink Scope block is added to the model to view signals during a simulation. xPC Target Scope blocks are added to the model for you to view signals while running the target application. Scope - Id1 displays the actual water level and the selected water level in the tank. Scope - Id2 displays the control signals. Both scopes are displayed on the target PC using a scope of type target. These scope blocks cannot be controlled by the Dials & Gauges Blockset or xPC Target interface blocks.

Note that the model `xpctank.mdl` does not contain any Dials & Gauges Blockset blocks or xPC Target interface blocks. This model is built entirely from standard Simulink blocks and Scope blocks from xPC Target. It does not differ in any way from a model you would normally use with xPC Target and it does not require any changes (except for tagging properties and signals) to use it with a Dials & Gauges Blockset interface.

## 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.

You do not have to modify this model when you use it with the Dials & Gauges Blockset, Virtual Reality Toolbox, or other third-party graphical elements.

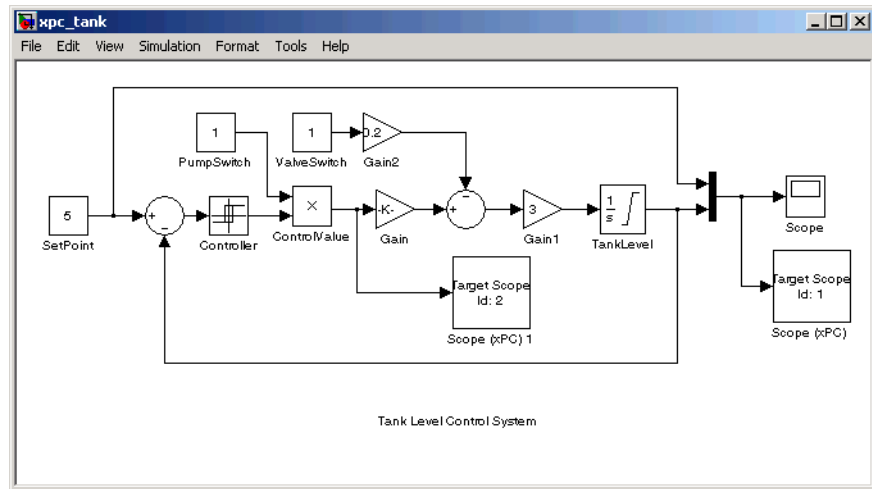
Creating a target application model is the first step you need to do before you can tag block parameters and block signals for creating a custom graphical interface:

- 1 In the MATLAB Command Window, type  
`xpctank`

A Simulink model for a water tank opens. This model contains a set of equations to describe the behavior of a water tank and a simple controller.

The controller regulates the water level in the tank. This model contains only standard Simulink blocks and you use it to create the xPC Target application.

- From the **File** menu, click **Save as** and enter a new filename. For example, enter `xpc_tank` and then click **OK**.



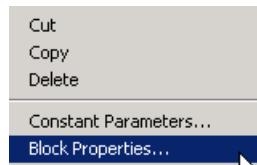
Your next task is to mark the block properties and block signals. See “Marking Block Parameters” on page 9-13 and “Marking Block Signals” on page 9-15.

### 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 (dials) in your user interface model.

After you create a Simulink model, you can mark the block parameters. This procedure uses the model `xpc_tank1.mdl` (or `xpctank.mdl`) as an example (see “Creating a Target Application Model” on page 9-12).

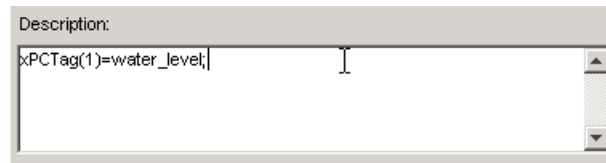
- Open a Simulink model. For example, in the MATLAB window, type `xpc_tank` or `xpctank`
- Point to a Simulink block, and then right-click.
- From the menu, click **Block properties**. (Do not click **Constant parameters**.)



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:

```
xPCTag(1, . . . index_n)= label_1 . . . label_n;
```

For single dimension ports, the following syntax is also valid:

```
xPCTag=label;
```

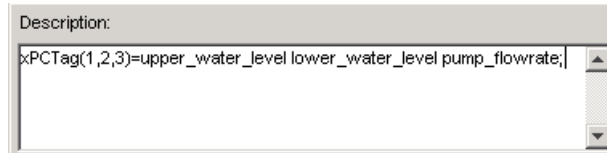
`index_n` — Index of a block parameter. Begin numbering parameters with an index of 1.

`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_1 label_2 label_3 label_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

```
xpc_tank1
```

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 9-15 and “Creating a User Interface Model” on page 9-19.

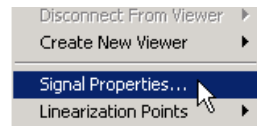
## Marking Block Signals

Tagging signals in your Simulink model allows the function `xpcsliface` to create From xPC Target interface blocks. These interface blocks contain the signals you connect to display devices (gauges) in your user interface model.

After you create a Simulink model, you can mark the block signals. This procedure uses the model `xpc_tank1.mdl` (or `xpctank.mdl`) as an example. See “Creating a Target Application Model” on page 9-12.

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 window, type `xpc_tank` or `xpc_tank1`
- 2 Point to a Simulink signal line, and then right-click.
- 3 From the menu, click **Signal properties**.

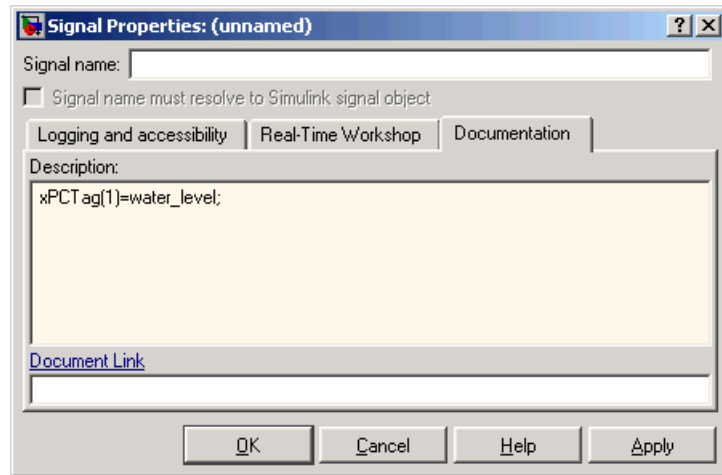


A **Signal properties** dialog box opens.

- 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;
```

- 6 From the **File** menu, click **Save as**. Enter a filename for your model. For example, enter

```
xpc_tank1
```

Your next task is to mark block parameters if you have not already done so, and then create the user interface template model. See “Marking Block Parameters” on page 9-13 and “Creating a User Interface Model” on page 9-19.

## Description of the User Interface Model

xPC Target includes the Simulink model `xpctank.mdl`. This is the model of a water tank with a pump, drain, and valve controller. See “Creating a Target Application Model” on page 9-12. Using Real-Time Workshop, xPC Target, and a third-party compiler, you can create a real-time target application from this model.

A user interface model is a Simulink user interface model you create as a graphical interface to a target application. You do this by using dial blocks, gauges blocks, and xPC Target interface blocks.

After you create a user interface template model with xPC Target interface blocks, you can create a user interface model with the following controls and displays:

- **Pump Switch Value** — Override the bang-bang controller by using a manual switch to turn the pump on and off. Turn the pump on or off by setting PumpSwitch to 1 or 0. Use a toggle switch.
- **Drain Valve Value** — Open or close the drain valve by setting ValveSwitch to 1 or 0. Use a toggle switch.
- **SetPoint Value** — Change the level of the water in the tank by changing the SetPoint block parameter **Constant value**. Use a slider to set this value.
- **TankLevel** — This is the water level and the output from the Integrator block TankLevel. Use a tank gauge to visualize this signal.
- **Controller OnOutputValue** — Change the pump flow rate by changing the Controller block parameter **Output when on**. Use a slider to set this value.

- **Controller OnSwitchValue/Controller OffSwitchValue** — Select the upper and lower water level by changing the Controller block parameters **Switch on point** and **Switch off point**. Use a slider to set these values. This single value adjusts the offset, which is the maximum and minimum change the water level is allowed above or below the set point before corrective action is taken by either turning the pump on or turning the pump off.

## Creating a User Interface Model

A user interface model contains components from the Dials & Gauges Blockset and xPC Target interface blocks. It functions as a graphical user interface that allows you to view signals with gauges and change parameters using dials.

After you tag block properties and block signals, you are ready to generate the user interface template model containing xPC Target interface blocks. These blocks connect your user interface model with your target application.

This procedure uses the Simulink model `xpc_tank1.mdl` as an example, and assumes you have tagged the block properties and block signals (see “Marking Block Parameters” on page 9-13 and “Marking Block Signals” on page 9-15).

- 1 In the MATLAB window, type

```
xpc_tank1
```

A window with the Simulink model `xpc_tank1.mdl` opens.

- 2 Type

```
xpcsliface('xpc_tank1')
```

A new Simulink window opens with the user interface template model.

- 3 From the **Simulation** menu, click **Configuration Parameters**.

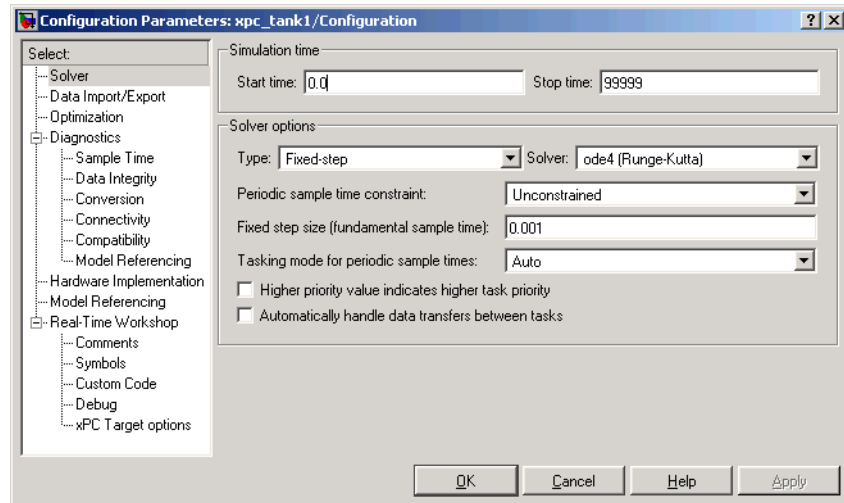
The **Configuration Parameters** dialog is displayed.

- 4 In the left pane, click the **Solver** node.

- 5 Simulink displays the **Solver** pane. This pane defines the initial stop and sample time for your target application.

- 6 In the **Stop time** box, enter `inf`. From the **Type** list, select **Fixed-step**. From the solver list, select **discrete** (no continuous states). In the **Fixed step size** box, enter `auto`.

Your **Solver** pane should look similar to the one shown below.

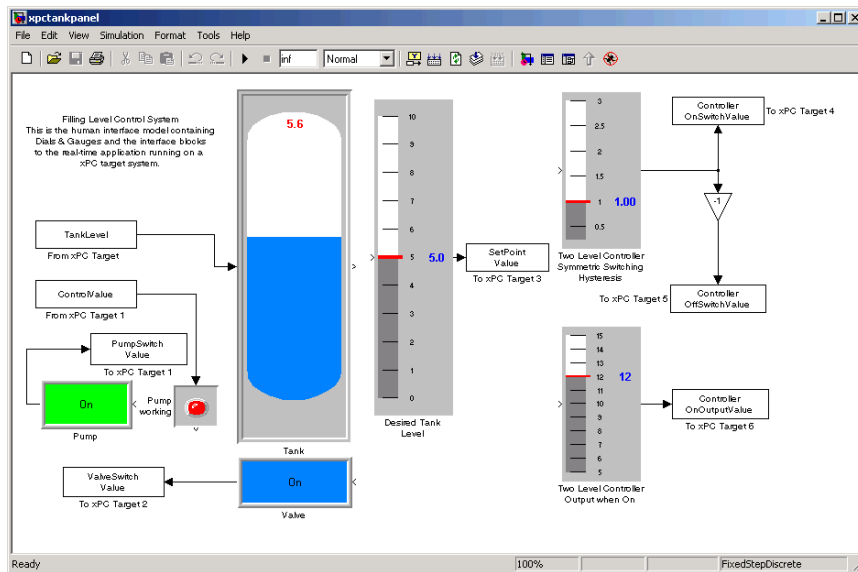


**Note** The user interface model should only contain discrete-time elements, and therefore you should only select the discrete solver. The model should not contain any continuous-time states.

Your next task is to add dials and gauges and connect them to the xPC Target interface blocks. See “Adding Dials & Gauges Blockset” on page 9-20.

## Adding Dials & Gauges Blockset

Open the model `xpctankpanel.mdl` for an example of connecting dials and gauges to xPC Target interface blocks. Your model should look similar to the figure shown below.



If you do not see the dials and gauges in the Active X blocks, try registering the Active X controls. In the MATLAB window, type

```
dng_register_ocx
```

Your next tasks are to create the target application from a target application model, and then run that application with the user interface model. See “Creating a Target Application” on page 9-21 and “Running a Target Application with a User Interface Model” on page 9-23.

## Creating a Target Application

Use this procedure to create a target application that you want to connect to a Simulink user interface model as a graphical interface with dial and gauge blocks.

After you create a Simulink model and tag the block parameters and block signals for creating xPC Target interface blocks, you can create a target application and download it to the target PC. This procedure uses the Simulink model `xpc_tank1.mdl` (or `xpctank.mdl`) as an example (see “Creating a Target Application Model” on page 9-12).

**1** Start or reset the target PC with an xPC Target boot disk in the 3.5 inch disk drive.

**2** In the MATLAB window, type  
xpc\_tank1 or xpctank

**3** From the **Simulation** menu, click **Configuration Parameters**.

The **Configuration Parameters** dialog is displayed.

**4** In the left pane, click the **Real-Time Workshop** node.

The **Real-Time Workshop** pane opens.

**5** In the **Target selection** section, click the **Browse** button at the **RTW system target file** list. Click `xpctarget.tlc`, and then click **OK**.

The system target file `xpctarget.tlc`, the template makefile `xpc_default_tmf`, and the make command `make_rtw` are automatically entered into the page.

**6** In the left pane, click the **Solver** node.

The **Solver** pane is displayed.

**7** Check that the **Stop time** is long enough for you to interact with the target application.

**8** Click **OK**.

**9** From the **Tools** menu, point to **Real-Time Workshop**, and then click **Build model**.

Real-Time Workshop, xPC Target, and a third-party C compiler create the target application and download it to the target PC.

Your next task is to start running the target application in real time and start a simulation of the user interface model. See “Running a Target Application with a User Interface Model” on page 9-23.

## Running a Target Application with a User Interface Model

After you create an xPC Target application, download that application to the target PC, and create a Simulink user interface model, you are ready to use the user interface model as a graphical interface to the target application (see “Creating a Target Application Model” on page 9-12 and “Creating a User Interface Model” on page 9-19).

**1** In the MATLAB window, type  
`xpc_tank1_gui` or `xpctankpanel`

**2** Type  
`start(tg)` or `+tg`

The target application starts running in real time on the target PC.

**3** In the Simulink window for the user interface model, from the **Simulation** menu, click **Normal**, and then click **Start**.

The user interface model begins a simulation in nonreal time. You should observe graphical information made available through the Dials & Gauges Blockset components. For example, the tank icon shows the water level rising.

When starting the simulation of your GUI interface, you might get an error with one or more of the To xPC Target or From xPC Target blocks. To correct this problem, create a temporary GUI using the function `xpcsliface('xpc_tank1')`, delete the problem blocks, and then copy new xPC Target interface blocks from the temporary GUI. Save the model and restart the simulation.

**4** Click the pump switch or drain valve buttons, or click and drag the slider knobs to change parameters to new values.





# xPC Target Web Browser Interface

---

xPC Target 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. This chapter includes the following section:

Web Browser Interface (p. 10-2)

Connect a target application running on a target PC to any host PC connected to a network

## Web Browser Interface

xPC Target has a Web server built into the kernel that allows you to interact with your target application using a Web browser. If the target PC is connected to a network, you can use a Web browser to interact with the target application from any host PC connected to the network.

Currently Microsoft Internet Explorer (Version 4.0 or later) and Netscape Navigator (Version 4.5 or later) are the only supported browsers.

This section includes the following topics:

- “Connecting the Web Interface Through TCP/IP” on page 10-2
- “Connecting the Web Interface Through RS-232” on page 10-3
- “Using the Main Pane” on page 10-6
- “Changing WWW Properties” on page 10-9
- “Viewing Signals with a Web Browser” on page 10-10
- “Viewing Parameters with a Web Browser” on page 10-11
- “Changing Access Levels to the Web Browser” on page 10-11

### Connecting the Web Interface Through TCP/IP

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

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

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

- 1 In the MATLAB window, type

```
xpcwwwenable
```

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

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

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

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

## Connecting the Web Interface Through RS-232

If the host PC and target PC are connected with a serial cable instead of a network cable, you can still connect the target application on the target PC to a Web browser on the host PC. xPC Target includes a TCP/IP to RS-232 mapping application. This application runs on the host PC and writes whatever it receives from the RS-232 connection to a TCP/IP port, and it writes whatever it receives 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 PC, you must load a target application onto the target PC. The target application does not have to be running, but it must be loaded. Also, your Web browser must have JavaScript and StyleSheets turned on.

- 1 In the MATLAB window, type

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

MATLAB is disconnected from the target PC, leaving the target PC ready to connect to another client. The TCP/IP stack 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 PC.

- 2 Open a DOS command window, and enter the command to start the TCP/IP to RS-232 gateway. For example, if the target PC 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 10-5.

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 2002 The MathWorks                 *  
*-----*  
Connecting COM to TCP port 22222  
Waiting to connect
```

If you did not close the MATLAB to target application connection, then 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
```

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

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

```
Error in ==>
C:\MATLABR13\toolbox\rtw\targets\xpc\xpc\@xpc\xpc.m
On line 31 ==> sync(xpcObj);
```

You must close MATLAB 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	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.  If you do not use this option, only the host PC that is connected to the target PC with a serial cable can connect to the selected port. For example, if you start the gateway on your host PC, with the default ports, you can type in the Web browser <code>http://localhost:2222</code> . However, if you try to connect to <code>http://Domainname.com:22222</code> , you will probably get a connection error.

<b>Command-Line Option</b>	<b>Description</b>
-t <i>tcpPort</i>	Use TCP port <i>tcpPort</i> . Default <i>t</i> is 22222. For example, to connect to port 20010, type -t 20010.
-h	Print a help message.
-c <i>comPort</i>	Use COM port <i>comPort</i> ( $1 \leq \textit{comPort} \leq 4$ ). 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**.

**System Status**

Application	xpcosc
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

SampleTime

StopTime

Apply Reset

**Navigation**

Scopes

Signals

Parameters Refresh

Screen Shot

**WWW Properties**

Maximum Signal Width

Refresh Interval

Click any button on the left to navigate

After you connect a Web browser to the target PC, 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 PC. 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 PC, the **Status** line is changed from **Stopped** to **Running**, and the **Start Execution** button text changes to **Stop Execution**.

- 3 Update the execution time and average task execution time (TET). Click the **Refresh** button. To stop the target application, click the **Stop Execution** button.
- 4 Enter new values in the **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"/>
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.

- 5 Select scopes to view on the target PC. 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 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 PC.

---

## Changing WWW Properties

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

- 1 In the **Maximum Signal Width** 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 PC 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 MATLAB 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 PC 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 PC, 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, press the **Edit** button to view the vector or matrix (in the correct shape). 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 **Real-Time Workshop** node.

The **Real-Time Workshop** pane opens.

- 3 In the **Target selection** section, access levels are set in the **RTW 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.

# Interrupts Versus Polling

---

xPC Target interrupt mode is the default real-time execution mode for the xPC Target kernel. For performance reasons, you might want to change the real-time execution mode to polling mode. This chapter includes the following section:

Polling Mode (p. 11-2)

Use polling mode as an alternative to interrupt mode for reducing latency times with I/O drivers

## Polling Mode

A good understanding of polling mode will help you to use it effectively, and a better understanding of interrupt mode will help you to decide under which circumstances it makes sense for you to switch to the polling mode. This section includes the following topics:

- “xPC Target Kernel Polling Mode” on page 11-2
- “Interrupt Mode” on page 11-2
- “Polling Mode” on page 11-4
- “Setting the Polling Mode” on page 11-6
- “Restrictions Introduced by Polling Mode” on page 11-8
- “Controlling the Target Application” on page 11-12
- “Polling Mode Performance” on page 11-13

### xPC Target Kernel Polling Mode

Polling mode for the xPC Target real-time 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 xPC Target.

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  $200 \text{ kHz}$ )

### Interrupt Mode

Interrupt mode is the default real-time execution mode for the xPC Target 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 ensures 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 to properly handle 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 proper execution 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 additionally latency.

- **Interrupt handler content latency** — If a background task has been executing for a longer time, say in a loop, its needed data will be available in the cache. But as soon as an interrupt occurs and the interrupt service handler is executed, the data needed in the interrupt handler might no longer be in the cache, causing the CPU to reload it from slower RAM. This introduces additional latency. Generally, an interrupt reduces the optimal execution speed or introduces latency, because of its unpredictable nature.

The xPC Target real-time 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, a certain amount of headroom (at least 5%) is necessary, which can cause additional cache misses and therefore nonoptimal execution speed.

## Polling Mode

Polling mode for the xPC Target real-time kernel does not have the 8  $\mu$ 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:

- **Accurate measurements** — Because the counter counts up with the CPU clock rate (~1 GHz nowadays), the accuracy of time measurements even in the microsecond range is very high, therefore leading to very small absolute real-time errors.
- **No overflow** — Because the counter is 64 bits wide, in practical use overflow does not occur, which makes a CPU time expensive overflow handler unnecessary.
- **No latency** — The counter resides on the CPU. Reading the counter value can be done within one CPU cycle, introducing almost no 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 xPC Target, 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 xPC Target 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, ensure 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 correct numerical results.

- **Use the fastest available computer hardware** — Ensure that the CPU with the highest clock rate available is used for a given PC form factor. For the desktop form factor, this would mean a clock rate above 1 GHz; for a mobile application, e.g., using the PC/104 form factor, this would mean a clock rate above 400 MHz. Most of the time, you should use a desktop PC, because the highest clocked CPUs are available for this form factor only. Executing `xpcbench` at the MATLAB prompt gives an understanding about the best performing CPUs for xPC 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

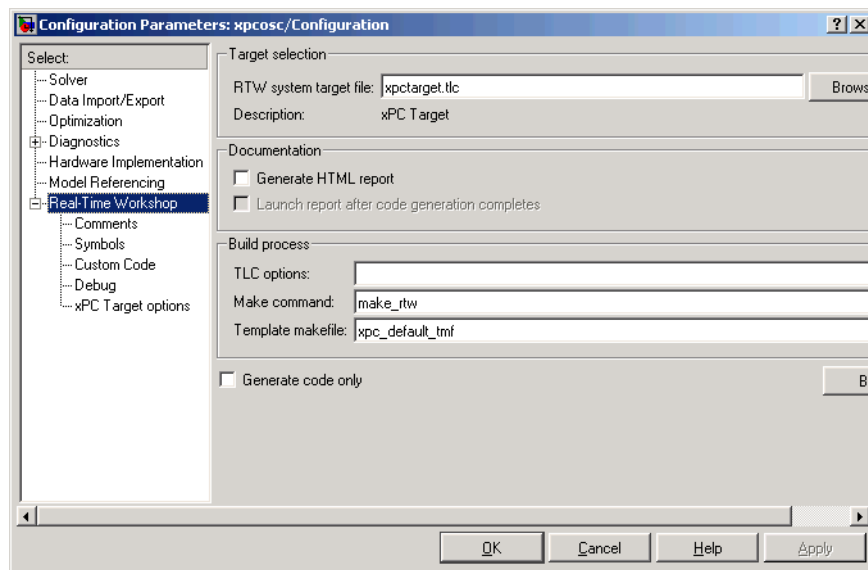
Polling mode is an alternative to the default interrupt mode of the real-time kernel. This means that the kernel on the bootable 3.5 inch disk created by the `xpcexplr` GUI allows running the target application in both modes without the necessity to use another boot disk.

By default the target application executes in interrupt mode. To switch to polling mode, you need to pass an option to the **RTW system target file** command. The following example uses `xpcosc.mdl`.

- 1 In the Simulink window, and from the **Tools** menu, point to **Real-Time Workshop**, and then click **Options**.

The **Configuration Parameters** dialog box opens.

- 2 In the left pane, click the **Real-Time Workshop** node.



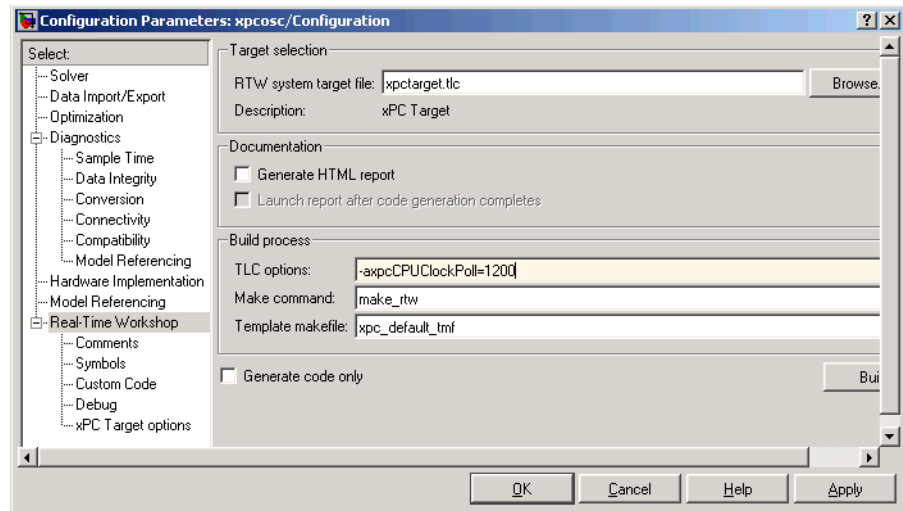
### 3 In the **TLC options** edit field, specify the option

`-axpcCPUclockPoll=CPUclockRateMHz`

The assignment of the clock rate of the target PC's CPU is necessary 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 an incorrect clock rate is provided, the target application executes at an incorrect base sample time. You can find out about the CPU clock rate of the target PC by rebooting the target PC and checking the screen output during BIOS execution time. The BIOS usually displays the CPU clock rate in MHz right after the target PC has been powered up.

For example, if your target PC is a 1.2 GHz AMD Athlon, specify option

`-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 checking for the correct 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, the background tasks, mainly the ones for host-target communication and target screen updating, are inactive. This is because all interrupts of the target PC are fully disabled during the execution of the target application. On one hand this ensures the highest polling performance; on the other hand, as a consequence the background tasks are not serviced.

Here is a list of 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 11-12.

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 xPC Target are not available in polling mode. These are

- Parameter tuning — Neither through the command-line interface nor through External mode
- Signal tracing through scope objects — Neither through scope objects of type host (`xpcscope` GUI or scripts) or type target (scopes on the target screen if property **Enable target scope** is enabled)
- Signal monitoring — You cannot run a GUI interface on the host PC 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 PC**

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 **Format -> Sample time color** feature of Simulink 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 Real-Time Workshop User's 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 in order to run properly, for example, for the detection of timeouts. Because the standard timing engine of the real-time kernel is not running during the entire target application execution in polling mode, timing information passed back to the drivers is incorrect. Therefore, you cannot use drivers importing 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 MathWorks Data Acquisition Toolbox to drive the corresponding TTL output pin of the host board to stop the target application execution from within MATLAB.

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 introduction of an additional 1  $\mu$ s latency necessary 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, which is the same machine that is at the top of the list displayed by `xpcbench`.

The minimum achievable base sample time for model `Minimal` (type `help xpcbench` in the MATLAB Command Window for further information) is 1  $\mu\text{s}$  with signal logging disabled and 2  $\mu\text{s}$  with signal logging enabled.

The minimum achievable base sample time for model `f14` (type `help xpcbench` for further information in the MATLAB window) using an `ode4` fixed-step integration algorithm is 4  $\mu\text{s}$  with signal logging disabled and 5  $\mu\text{s}$  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  $\mu\text{s}$  with signal logging disabled and 12  $\mu\text{s}$  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  $\sim 28 \mu\text{s}$  or a sample rate of  $\sim 33 \text{ kHz}$ . For this application, the overall performance increase using polling mode is almost a factor of 3.



# xPC Target and Fortran

---

xPC Target supports the incorporation of Fortran code into Simulink models. This chapter describes the following:

Introduction (p. 12-2)

Use Simulink S-functions to incorporate Fortran code into xPC Target.

Step-by-Step Example of Fortran and xPC Target (p. 12-5)

Follow the example to build your own xPC Target application with Fortran code.

## Introduction

xPC Target supports Fortran in Simulink models with S-functions. (See “Creating Fortran S-Functions” in the Writing S-Functions documentation for a description of how to incorporate Fortran code into Simulink models.) This chapter describes how to incorporate Fortran into a Simulink model for xPC Target.

This section has the following topics:

- “Simulink Demos Directory” on page 12-2
- “Prerequisites” on page 12-3
- “Steps to Incorporate Fortran in Simulink for xPC Target” on page 12-3

The example below uses one of the provided Fortran demo files, Atmosphere model.

### Simulink Demos Directory

The Simulink demos directory 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  
demos

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

- 2** From the left side of the window, select **Simulink**, then **Features**.

A list of Simulink examples appears on the right side of the window.

- 3** Click **Custom Code and Hand Coded Blocks: M, C/C++, Fortran, etc.**.

The associated Simulink demos page opens.

- 4** Click **Open this model**.

A library of S-function examples is displayed.

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

A library of Fortran S-functions and associated templates appears. This library also contains a README block. This file contains the same information as that contained in “Creating Fortran S-Functions” in the Writing S-Functions documentation. In that chapter, the sections “Creating Level 2 Fortran S-Functions” and “Porting Legacy Code” are most applicable to xPC Target.

## Prerequisites

You must have the following to use Fortran for xPC Target applications:

- xPC Target Version 1.3 or later
- Compaq Visual Fortran Compiler Version 6.5 or later

## Steps to Incorporate Fortran in Simulink for xPC Target

This section 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 C-MEX wrapper S-function for Simulink. 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 DLL.

- 4 Run a simulation C-MEX file with Simulink to validate the compiled Fortran code and wrapper S-function.
- 5 Copy relevant Fortran run-time libraries to the application build directory for the xPC Target application build.

- 6 Define the Fortran libraries, and the Fortran object files from step 1, in the **Real-Time Workshop** dialog 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 Real-Time Workshop build procedure for the demo model. Real-Time Workshop builds and downloads the xPC Target onto the target PC.

## Step-by-Step Example of Fortran and xPC Target

This example uses the demo Atmosphere model that comes with Simulink. The following procedures require you to know how to write Fortran code appropriate for Simulink and xPC Target. See “Creating Fortran S-Functions” in the Writing S-Functions documentation for these details.

This section includes the following topics:

- “Creating an xPC Target Atmosphere Model for Fortran” on page 12-5
- “Compiling Fortran Files” on page 12-7
- “Creating a C-MEX Wrapper S-Function” on page 12-9
- “Compiling and Linking the Wrapper S-Function” on page 12-9
- “Validating the Fortran Code and Wrapper S-Function” on page 12-10
- “Preparing the Model for the xPC Target Application Build” on page 12-11
- “Building and Running the xPC Target Application” on page 12-13

Before you start, you should create an xPC Target Simulink model for the Atmosphere model. See “Creating an xPC Target Atmosphere Model for Fortran” on page 12-5.

### Creating an xPC Target Atmosphere Model for Fortran

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

**1** From the MATLAB window, change directory to the working directory, for example, `xpc_fortran_test`.

**2** Type

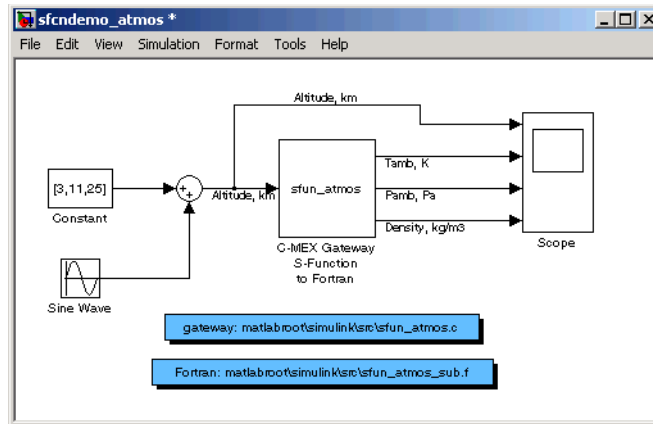
```
sfcdemo_atmos
```

The `sfcdemo_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 Click **Apply**, then **OK**.
- 11 For the **Sample time** parameter, enter 0.05.
- 12 Click **Apply**, then **OK**.
- 13 From the **File** menu, click **Save as**. Browse to your current working directory, 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 12-7.



## Compiling Fortran Files

This section describes the ways that you can compile Fortran code for xPC Target. Choose the procedure most convenient to your needs:

- DOS command window
- Microsoft Developer Studio IDE

Before you start,

- 1 Change directory to <MATLAB root>\simulink\src.
- 2 Copy the file `sfun_atmos_sub.f` into your Fortran working directory, for example, `xpc_fortran_test`.

This is the sample Fortran code that implements a subroutine for the Atmosphere model.

Your next task is to compile the Fortran code for xPC Target. See “DOS Command Line” on page 12-7 or “Microsoft Developer Studio IDE” on page 12-8.

### DOS Command Line

This section describes the procedure for compiling Fortran files from a DOS command window:

- 1 Ensure that the system environment has the correct path and variable settings so that you can start the Fortran compiler from the DOS command prompt.
- 2 From the DOS prompt, change directory to the working directory, for example, `xpc_fortran_test`.

### 3 Type

```
f132 -c /iface:cref -G5 -Ox -Zi sfun_atmos_sub.f
```

This command generates the `sfun_atmos_sub.obj` file.

Of these options, `-c` and `/iface:cref` are the most important. The remaining options are typical compiler optimization and debug options.

The `-c` option ensures that the compiler compiles only the file and does not link it into an executable.

The `/iface:cref` option defines the interface as C, making direct calls of the subroutines from C code possible.

Your next task is to create a wrapper S-function. See “Creating a C-MEX Wrapper S-Function” on page 12-9.

### Microsoft Developer Studio IDE

This section describes how to use Microsoft Developer Studio IDE to compile Fortran code.

- 1 Define a Fortran project in the Microsoft Developer Studio IDE. This procedure lets the IDE handle the compilation process.
- 2 Specify at least the `/iface:cref` and `-c` options for the developer studio.

The final outcome should be the `sfun_atmos_sub.obj` file.

For more information on the Microsoft Developer Studio IDE, refer to the Fortran compiler documentation.

Your next task is to create a C-MEX wrapper S-function. See “Creating a C-MEX Wrapper S-Function” on page 12-9.

## Creating a C-MEX Wrapper S-Function

This section assumes that you have compiled your Fortran code. See “Compiling Fortran Files” on page 12-7.

Write the wrapper S-function for `sfun_atmos_sub.f`. A wrapper S-function is code that incorporates existing Fortran code into a Simulink S-Function block. For details on writing such wrapper functions, refer to the Writing S-Functions Simulink documentation.

The wrapper S-function calls the Fortran subroutine `Atmos` with the appropriate calling convention: `atmos_`. (Refer to “Creating Fortran S-Functions” in the Writing S-Functions documentation for further information about calling conventions.) The wrapper S-function file for this example is called `sfun_atmos.c`.

- 1 Change directory to `<MATLAB root>\simulink\src`.
- 2 Copy the file `sfun_atmos.c` into your Fortran working directory, 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 12-9.

## Compiling and Linking the Wrapper S-Function

Create (compile and link) a DLL (C-MEX DLL) from the `sfun_atmos.c` file. Use the `mex` command with a C/C++ compiler such as Microsoft Visual C/C++ Version 6.0.

Before you start, copy the following run-time library files from the Fortran compiler installer directory, such as `C:\Program Files\Microsoft Visual Studio\DF98\lib`, into the working directory, `xpc_fortran_test`. Copying these files simplifies the build process.

- `dfor.lib`
- `dformd.lib`
- `dfconsol.lib`
- `dfport.lib`

This section assumes that you have created a C\_MEX wrapper S-function. See “Creating a C-MEX Wrapper S-Function” on page 12-9.

Invoking the `mex` command includes the following steps:

**1** Compile the wrapper C file `sfun_atmos.c`. Be sure to link in the following:

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

**2** Type

```
mex -v LINKFLAGS#"$LINKFLAGS dformd.lib dfconsol.lib  
dfport.lib" sfun_atmos.c sfun_atmos_sub.obj
```

Ensure that this whole command is all on one line. This command compiles and links the `sfun_atmos_sub.c` file. It creates the `sfun_atmos.dll` file in the same directory.

---

**Note** If this command generates a conflict error with `libc`, you might need to add the option `/NODEFAULTLIB:libc.lib` to the command. For example, `mex -v /NODEFAULTLIB:libc.lib LINKFLAGS#"$LINKFLAGS dformd.lib dfconsol.lib dfport.lib" sfun_atmos.c sfun_atmos_sub.obj`.

---

Your next task is to validate the Fortran code and wrapper S-function. See “Validating the Fortran Code and Wrapper S-Function” on page 12-10.

## Validating the Fortran Code and Wrapper S-Function

Validate the generated DLL, `sfun_atmos.dll`. Bind the S-function DLL 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 section assumes that you have compiled and linked a wrapper S-function. See “Compiling and Linking the Wrapper S-Function” on page 12-9.

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

- 1 At the MATLAB window, type

```
fortran_atmos_xpc
```

This opens the Simulink model associated with the Atmosphere model. This model includes the correct S-function block that is bound to `sfun_atmos.dll`.

- 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 12-11.

## 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 12-9) for the link stage.

This section assumes that you have validated the Fortran code and wrapper S-function (see “Validating the Fortran Code and Wrapper S-Function” on page 12-10).

- 1 At 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 **Real-Time Workshop** node.

The **Real-Time Workshop** pane opens.

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

**5** Click `xpctarget.tlc`.

**6** In the **Make command** field, replace `make_rtw` with the following string:

```
make_rtw S_FUNCTIONS_LIB="..\sfun_atmos_sub.obj ..\dfor.lib  
..\dfconsol.lib ..\dfport.lib"
```

Ensure that this whole command is all on one line.

**7** Press **Apply**.

**8** Press **OK**.

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

This command requires that the application build directory be the current directory (one level below the working directory, `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 12-13.

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

This section 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 12-11.

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 `xpcexplr`.

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 12-10).





# Troubleshooting

---

This chapter describes guidelines, hints, and tips for issues you might encounter while using xPC Target. 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. This chapter includes the following sections:

General Troubleshooting Hints and Tips (p. 13-2)

General xPC Target troubleshooting hints and tips

Installation, Configuration, and Test Troubleshooting (p. 13-6)

General xPC Target troubleshooting guidelines for installation, configuration, and testing

Advanced Troubleshooting (p. 13-12)

Less common, more advanced xPC Target troubleshooting guidelines

## General Troubleshooting Hints and Tips

This section lists general troubleshooting tips that you can use as a first attempt to resolve your issues. This section has the following topics:

- “Is There Communication Between Your PCs?” on page 13-3
- “xPC Target and the Target PC BIOS” on page 13-4
- “What PCI Boards Are Installed on Your System?” on page 13-4
- “How to Get Updated xPC Target Releases” on page 13-4
- “Are You Working with a New xPC Target Release?” on page 13-5
- “What Does the Target PC Display?” on page 13-5
- “Refer to The MathWorks Support Web Site” on page 13-5
- “Refer to the Documentation” on page 13-5

### Is Your Host PC MATLAB Halted?

If your host PC MATLAB halts while creating an xPC Target boot disk,

- Use another formatted disk to create the xPC Target boot disk.
- If your host PC has anti-virus software, it might conflict with MATLAB. Disable the software while using MATLAB.
- Verify that the host PC 3.5 inch disk drive is accessible. If it is not accessible, replace the 3.5 inch disk drive.

### Is Your Target PC Unable to Boot?

If your target PC cannot boot with the xPC target boot disk,

- Use another formatted disk and create a new xPC Target boot disk.
- Verify that the current properties on the xPC Target boot disk correspond to the environment variables of xPC Target Explorer.
- Verify that the xPC Target boot disk contains files like the following:
  - BOOTSECT.RTT
  - checksum.dat
  - XPCTGB1.RTA

Note that the name of the last file varies depending on the communication method.

- If any of these files are not present, reinstall xPC Target. This should fix any corrupted files from the previous (initial) installation.
- If problems persist, see “Troubleshooting the Boot Process” in Chapter 3 of the xPC Target Getting Started documentation.
- If you still cannot boot the target PC, you might need to replace the target PC 3.5 inch disk drive.

### Is the Target PC Halted?

If your target PC displays a System Halted message while booting,

- Verify that the **TcpIp target driver** parameter is configured correctly in xPC Target Explorer, recreate the xPC Target boot disk, and use that new disk to boot the target PC.
- Ensure that xPC Target supports your target PC hardware. Be sure to verify the network communication hardware.

### Is There Communication Between Your PCs?

If you are using xPC Target in a host/target PC client/server setup, use the following MATLAB commands from the host PC:

- `xpctargetping`
- `xpctest`

The `xpctargetping` command performs a basic communication check between the host and target PC. This command only returns success if the xPC Target kernel is loaded and is running and the communication between host and target PC is working properly. Use this command for a quick check of the host PC/target PC communications.

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 PC is running an old xPC Target boot disk that is not in sync with the xPC Target release installed on the host PC. Create a new boot disk for each new release of xPC Target.
- The communication between the host PC and target PC is TCP/IP, set the host PC network interface card (NIC) card to half-duplex mode. Do not set the mode to full-duplex mode.

### **xPC Target and the Target PC BIOS**

The settings of your target PC BIOS will affect your xPC Target. As a general rule, ensure that the host and target PC BIOS have at least the following settings:

- RS-232 communication — If you are using RS-232 communications, ensure that COM ports are enabled for both host and target PCs. Also, ensure through the BIOS that COM1 has a base address of 3F8 and an IRQ of 4. COM2 must have a base address of 2F8 and an IRQ of 3. These are the default base address values. Do not change these values.
- Plug-and-Play (PnP) operating system — Disable this feature to ensure that the PCI BIOS sets up the plugged-in PCI cards properly. The xPC Target kernel is not a PnP operating system; you must ensure that this feature is disabled or PCI devices will not work on xPC Target.
- Power Saving modes — Disable all power saving modes.
- USB Support — Disable all USB support, including general USB and USB keyboard support. Failure to do this will cause occasional long task execution times (TET).
- PCI boards — Do not detect PCI boards with class code 0xff in target PC BIOS. Set to 0ff to enable the BIOS to detect and configure all boards.

### **What PCI Boards Are Installed on Your System?**

Use the `getxpcpci` MATLAB command to determine what PCI boards are installed in your xPC Target system. For example,

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

### **How to Get Updated xPC Target Releases**

- 1 Start Simulink -> xPC Target -> Product News (Web).**

- 2 Look for the section on downloading software and select the version you want.

### **Are You Working with a New xPC Target Release?**

If you are working with a new xPC Target release, either one you download from The MathWorks Web site ([http://www.mathworks.com/web\\_downloads/](http://www.mathworks.com/web_downloads/)) or one you install from a CD, you must do the following:

- At the MATLAB Command Window, invoke `xpcexplr`.
- Recreate your xPC Target environment (see “Serial Communication” or “Network Communication” in Chapter 2 of the Getting Started with xPC Target documentation).
- Create a new boot disk. Use a new 3.5 inch disk.
- Rebuild target applications on that new xPC Target release.

### **What Does the Target PC Display?**

From the host PC, you can view the target PC monitor with the MATLAB `xpctargetspy` command.

### **Refer to The MathWorks Support Web Site**

This chapter contains general xPC Target troubleshooting tips. Refer to the MathWorks Support xPC Target Web site (<http://www.mathworks.com/support/product/XP>) for more specific troubleshooting solutions. The xPC Target documentation is also available from this site.

### **Refer to the Documentation**

The xPC Target documentation has hints and tips embedded throughout. You should install the Help and PDF documentation to provide easy reference:

- The xPC Target Help documentation is available for installation when you install the xPC Target product either from the CD or Web download.
- The PDF documentation is available for installation from <http://www.mathworks.com>.

## Installation, Configuration, and Test Troubleshooting

The following are some issues you might encounter with xPC Target when trying to set up.

### Troubleshooting xpctest Results

This section assumes that you have read the “Testing and Troubleshooting the Installation” section of the Getting Started with xPC Target documentation.

**xpctest: Test 1 Fails.** First, perform the procedure described in the “Test 1, Ping Target System Standard Ping” section of the Getting Started with xPC Target documentation.

---

**Note** You can ignore this section if you are using a serial connection. Test 1 is skipped for serial connections.

---

If you are using a TCP/IP connection and need more help with Test 1, check the following:

- Be sure to use a supported Ethernet card on the target PC. The following Ethernet controllers are supported:
  - Intel 82559 based cards
  - Intel 82559ER based cards
  - Intel 82550 based cards
  - NE2000 based cards
  - AMD lance 79C971(RTLANCE) based cards
  - SMC91C9X based cards

See “Ethernet Chips Supported by xPC Target” in Chapter 2 of the Getting Started with xPC Target documentation for further details.

- Verify that your hardware is operating correctly. For example, check for faulty network cables and other hardware.
- If you run xpctest from an UNC network directory, such as \\Server\user\work, a workaround is to change the current MATLAB directory to a local directory and run the test again.

**xpctest: Test 2 Fails.** First, follow the procedure described in the “Test 2, Ping Target System xPC Target Ping” section of the Getting Started with xPC Target documentation.

If you need more help with Test 2, check the following:

- Use the PC MATLAB command `xpcexplr` to check the environment variables, in particular **Target PC IP address**. If Test 1 passes but Test 2 fails, you might have entered an incorrect IP address.
- If you have a TCP/IP connection, make sure you are using a supported Ethernet card (see “xpctest: Test 1 Fails” on page 13-6).
- For RS-232 connection,
  - Use a null modem cable (see the “Hardware for Serial Communication” section of the Getting Started with xPC Target documentation). If you do not use a null modem cable for an RS-232 connection, communication between the host and target PCs will fail. A null modem cable is shipped with xPC Target.
  - If you do have a null modem cable, check the COM ports on the host and target PC. For example, ensure that the ports are enabled, you have connected the appropriate COM port, and the COM port matches that specified for `xpcexplr`.
  - Ensure that the COM ports on the host and target PCs are enabled in the BIOS. If they are disabled, Test 2 fails.

**xpctest: Test 3 Fails.** First, follow the procedure described in the “Test 3, Reboot Target Using Direct Call” section of the Getting Started with xPC Target documentation.

If you need more help with Test 3, check the following:

- Did you get the following error?
  - `ReadFile Error: 6`Older xPC Target releases might receive this error. This message might occur if the host PC initiates communication with the target PC while the target PC is rebooting, but the kernel on the target PC has not yet loaded. As a workaround, run `xpctest` with the `noreboot` option. For example,

```
xpctest noreboot
```

This command runs the test without trying to reboot the target PC. It displays the following message:

```
### Test 3, Reboot target using direct call: ... SKIPPED
```

- If you directly or indirectly modify the xpcosc demo mode that is supplied with xPC Target, Test 3 is likely to fail. To pass this test, restore the original xpcosc demo model using one of the following methods:
  - (Preferred) Download a new copy of the model from the MathWorks FTP site ([ftp://ftp.mathworks.com/pub/tech-support/xpcosc\\_model/](ftp://ftp.mathworks.com/pub/tech-support/xpcosc_model/)). Overwrite the old xpcosc model with this new one in the directory `matlabroot\toolbox\rtw\targets\xpc\xpcdemos`
  - Recreate the original model.

---

**Note** Do not modify any of the files that are installed with xPC Target. If you want to modify one of these files, copy the file and modify the copy.

---

**xpctest: Test 4 Fails.** First, follow the procedure described in the “Test 4, Build and Download Application” section of the Getting Started with xPC Target documentation.

If you need more help with Test 4, check the following:

- Verify that a supported compiler is being used.
- If the communication between the host PC and target PC is TCP/IP, set the host PC network interface card (NIC) card to half-duplex mode. Do not set the mode to full-duplex mode.
- Verify the specified path to the supported compiler. You need only the root path to the compiler, not the full path. If you incorrectly specify a path, you might get the following error:

```
Error executing build command: Error using ==> make_rtw
Error using ==> rtw_c (SetupForVisual)
Invalid DEVSTUDIO path specified
```

or the following error:

```
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.
```

To correct these errors,

- 1 Ensure that your compiler is properly installed. For example, all Microsoft Visual compiler components must be in the Microsoft Visual Studio folder after installation.
- 2 At the MATLAB prompt, type `xpcexplr`. For example,  
`xpcexplr`
- 3 In the **Select C compiler** field, select the appropriate compiler type (VisualC or Watcom).
- 4 In the **Compiler Path** field, enter the root path to the compiler. For example  
`d:\applications\microsoft visual studio`

Do not add a terminating backslash (\) at the end of the path.

**xpctest: Test 5 Fails.** This error occurs only when the environment variable settings are out of date.

To correct this, perform the following:

- 1 At the MATLAB prompt, type `xpcexplr`. For example,  
`xpcexplr`
- 2 Inspect the environment variables.
- 3 If you have xPC Target Embedded Option installed, ensure that in the **xPC Target boot mode** section, you have selected the **BootFloppy** option.
- 4 Make necessary changes.
- 5 Recreate the boot disk. Perform the following:

- a Insert a formatted floppy disk.
  - b Click **BootDisk**.
- 6 Reboot the target PC.
- 7 Rerun `xpctest`.

If this procedure does not resolve the issue, perform the following:

- 1 At the MATLAB command line, type `updatexpcenv`. For example,  
`updatexpcenv`
- 2 Recreate the boot disk. Perform the following:
  - a Insert a formatted floppy disk.
  - b At the MATLAB window, type  
`xpcbootdisk`
- 3 Reboot the target PC.
- 4 Rerun `xpctest`.

**xptest: Test 6 Fails.** This test runs the basic target object constructor, `xpc`. This error rarely occurs without an earlier test failing.

To correct this, perform the following,

- 1 At the MATLAB command line, refer to and read the `xpc` reference page. For example,  
`help xpc`
- 2 Follow any guidance that might be helpful.
- 3 Reboot the target PC.
- 4 Rerun and check the results of earlier tests and make the necessary corrections.

**xptest: Test 7 Fails.** This test executes a target application (xpcosc) on the target PC. This test will fail if you change the xpcosc model start time to something other than 0, such as 0.001. This change causes the test, and MATLAB itself, to halt. To correct this, set the xpcosc model start time back to 0.

**xptest: Test 8 Fails.** This test executes a target application (xpcosc) on the target PC. This test might fail if you change the xpcosc model (for example, if you remove the output block).

To correct this, perform one of the following:

- Set the model back to the original configuration.
- Download a new copy of the model from the MathWorks FTP site ([ftp://ftp.mathworks.com/pub/tech-support/xpcosc\\_model/](ftp://ftp.mathworks.com/pub/tech-support/xpcosc_model/)).
- Overwrite the old xpcosc model with this new one in the directory

```
matlabroot\toolbox\rtw\targets\xpc\xpcdemos
```

Other issues might also cause this test to fail. If you still need more help, check the following:

- There is a known issue with an earlier version of xPC Target (1.3). It might occur when you run xptest two immediately consecutive times. See the known issue and solution documented in: <http://www.mathworks.com/support/solutions/data/27889.html>.
- If you are running a new xPC Target release, be sure that you have a new boot disk for this release. See “Are You Working with a New xPC Target Release?” on page 13-5.
- If you are installing another version of xPC Target 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 Target Simulink block library pop-up window. If the version number is not the one to which you want to upgrade, reinstall xPC Target.

## Advanced Troubleshooting

The following are some advanced issues or questions you might have with xPC Target. This section has the following topics:

- “General I/O Troubleshooting Guidelines” on page 13-12
- “xPC Target and BIOS” on page 13-13
- “Attempts to Run Any Model Cause CPU Overload Messages on Target PC” on page 13-13
- “Building a Model That Contains a CAN Board” on page 13-14
- “Obtaining PCI Board Slot and Bus Information” on page 13-15
- “Why Does xPC Target Lose Connection with the Host PC When Downloading Some Models?” on page 13-16
- “Why Is My Requested xPC Target Sample Time Different from the Measured Sample Time?” on page 13-18
- “Why Did I Get Error -10: Invalid File ID on the Target PC?” on page 13-20
- “Can I Write Custom xPC Target Device Drivers?” on page 13-20
- “Can I Create a Stand-Alone xPC Target Application to Interact with a Target Application?” on page 13-21
- “Can Signal Outputs from Virtual Blocks Be Tagged?” on page 13-21
- “Why Has the Stop Time Changed?” on page 13-21
- “Why Do I Get File System Disabled Error?” on page 13-22
- “How can I Diagnose Network Problems with xPC Target?” on page 13-22

### General I/O Troubleshooting Guidelines

If you encounter issues using the xPC Target I/O drivers,

- Ensure that you have properly configured the driver.
- Ensure that you are using the latest version of xPC Target.
- Test the hardware using the available diagnostic software included with the I/O board from the manufacturer.
- Try a different target PC to verify the behavior.
- Report the issue to The MathWorks Support at ([http://www.mathworks.com/support/contact\\_us/index.html](http://www.mathworks.com/support/contact_us/index.html)).

## xPC Target and BIOS

The settings of your host or target PC BIOS will affect your xPC Target results. See “xPC Target and the Target PC BIOS” on page 13-4 for recommended settings. Incorrect BIOS settings can cause questions like the following:

- getxpcpci can detect PCI boards, but autosearch -1 cannot
- Why can my stand-alone application run on some target PCs, but not others?
- Why is my target PC crashing while downloading applications?
- Why is my target PC104 hanging on boot up?
- Why is my boot time slow?
- Why is my software not running in real time?

## Attempts to Run Any Model Cause CPU Overload Messages on Target PC

This error might occur if you have

- A model sample time that is too small (see “Dealing with Small Model Sample Times” on page 13-13)
- Enabled Advanced Power Management, USB ports in the target PC BIOS, or Plug and Play (PnP) (see “Target PC BIOS” on page 13-14)

**Dealing with Small Model Sample Times.** If the model has too small a sample time, a CPU overload can occur. This error indicates that to run the target application, the model sample time property requires more CPU 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 correct the issue, perform one of the following:

- Change the model Fixed step size property to a larger value and rebuild the model. Use the **Solver** node in the Simulink model **Configuration Parameters** dialog.
- Run the target application on a target PC with a faster processor.

**Target PC BIOS.** This section assumes that the target PC has BIOS running on it. A CPU overload error can occur if any of the following are enabled:

- Advanced Power Management
- USB ports
- Plug and Play (PnP)

Enabling any of these properties causes non-real-time behavior from the target PC. You must disable these BIOS properties for the target PC to run the target application properly in real time.

**More Help.** If the preceding procedures do not solve your issue,

- Run `xpcbench` at the MATLAB command line. For example,  
`xpcbench('this')`

This program accurately evaluates your system. The results indicate the smallest base sample time that an xPC Target application can achieve on your system. For more information on `xpcbench`, type `help xpcbench` at the MATLAB prompt or

<http://www.mathworks.com/support/product/XP/productnews/benchmarks.html>.

- Set up the xPC Target environment with a different target PC. Compare the result with the original target PC.

### **Building a Model That Contains a CAN Board**

If you want to use the target PC in a CAN network, you must remember to set up the xPC Target environment for a CAN library. If you do not configure a CAN library into the system, you will get CAN errors when building the target application.

To correct the issue,

- 1 At the MATLAB command line, type `xpcexplr`. For example,  
`xpcexplr`

The **xPC Target Explorer** window is displayed.

- 2 At the **CAN Library** parameter, select the appropriate CAN library for the board in your system.

- 3 Create a new boot disk.
- 4 Reboot the target PC with the new boot disk.
- 5 Rerun `xpctest`.
- 6 To update the model with the new information, press **Ctrl+D**.
- 7 Build the model again.

**More Help.** If the preceding procedures do not solve your issue,

If you can build a target application with the CAN board in your model, but cannot download that application to the target, ensure that

- You are using a supported CAN board.
- You selected the correct choice from the **Can Library** parameter in `xpcexplr`.

### Obtaining PCI Board Slot and Bus Information

This section describes how to obtain information about the PCI devices in your xPC Target system. This information is useful if you have or want to use multiple boards of a particular type in your system. Before you start, ensure that the I/O drive supports multiple boards. Refer to one of the following:

- xPC Target I/O Reference documentation
- 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 the board type supports multiple boards, and these boards are installed in the xPC Target system, perform the following procedure to obtain the bus and slot information for these boards:

- 1 For example, type  

```
getxpcpci('all')
```
- 2 Note the contents of the Bus and Slot columns of the PCI devices in which you are interested.

- 3** 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 the xPC Target I/O Reference documentation.

### **Why Does xPC Target Lose Connection with the Host PC When Downloading Some Models?**

If you are using xPC Target hardware in a model, downloading the model might cause a loss of communication between the target PC and host PC if either of the following is true:

- The referenced xPC Target board has an initialization time that is too slow.
- The referenced xPC Target driver has an issue.

**xPC Target I/O Boards with Slow Initialization Times.** Some xPC Target boards have an initialization time that is too slow. This might cause xPC Target to run out of time before a model downloads, causing the host PC to disconnect from the target PC.

By default, if the host PC does not get a response from the target PC after downloading a target application and waiting 5 seconds, the host PC software times out. The target PC 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 sufficient. 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 PC 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 enact a more permanent resolution and increase the timeout value using the following procedure:

- 1 In your MATLAB working directory, create a file called `xpcdltimeout.dat`.
- 2 In this file, put a single integer. For example, enter

```
20
```

In this case, the host PC waits for about 20 seconds before declaring that a timeout has occurred. Note that it does not take 20 seconds for every download. The host PC polls the target PC about once every second, and if a response is returned, declares success. Only in the case where a download really fails does it take the full 20 seconds.

If the file `xpcdltimeout.dat` exists, it is read once before every download. To change the timeout value, change the number in this file. Setting the timeout to 5 is the same as the default. Note also that simply removing the file does not change the timeout to the default value. xPC Target uses the last value you entered until you restart MATLAB.

**xPC Target Driver Software Issues.** If there really is an error in a driver that causes xPC Target 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 PC and target PC.

To get yourself back up and running,

- 1 Remove the reference to the problem driver from the model.
- 2 Reboot the target PC.
- 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 The 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 MATLAB.

### **Why Is My Requested xPC Target 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. For xPC Target, 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. It is this limitation that 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 xPC Target, 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, xPC Target calculates that number as

$$100 \times 10^{-6} \text{ seconds} \times 1.193 \times 10^6 \text{ ticks/seconds} = 119.3 \text{ ticks}$$

xPC Target 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 .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 .29% (.145/50). Compared to the previously calculated value of .25%, there is a difference of .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 .019, which deviates from the expected signal value by .038% (.019/50.002). The deviation when the sample time is 1/10000 is .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 Windows or Linux, occasionally insert extra long intervals to compensate for errors in the timer. Be aware that xPC Target does not attempt to compensate for timer errors. This is because for xPC Target, 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 .5 microsecond from interrupt to interrupt.

---

### **Why Did I Get Error -10: Invalid File ID on the Target PC?**

You might get this error if you are acquiring signal data with a scope of type file. This error occurs because the size of the signal data file exceeds the available space on the disk. The signal data will most likely be corrupted 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 Write Custom xPC Target Device Drivers?**

You might want to write your own driver if you want to include an unsupported device driver in your xPC Target system. The xPC Target documentation does not currently describe how to write your own xPC Target device drivers. Refer to the following for further information:

- <http://www.mathworks.com/support/solutions/data/31528.html>
- Existing xPC Target device driver source code. Refer to the following directory:  
    `MATLABROOT\toolbox\rtw\targets\xpc\target\build\xpcblocks`
- In the include directory of the device driver source code area, pay particular attention to the following files:
  - `io_xpcimport.h`
  - `pci_xpcimport.h`
  - `time_xpcimport.h`

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 xPC Target, you must develop drivers with minimal latency. And since most drivers access the I/O hardware at the lowest possible level (register programming), you must have a good understanding of how to control the board with register information. Indirectly, this means that you must 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.

### **Can I Create a Stand-Alone xPC Target Application to Interact with a Target Application?**

Yes. You can use either the xPC Target API dynamic link library (DLL) or the xPC Target component object model (COM) API library to create custom stand-alone applications to control a real-time application running on the target PC. To deploy these stand-alone applications, you must have the xPC Target Embedded Option. Without the xPC Target Embedded Option, you can create and use the stand-alone application in your environment, but cannot deploy that application another host PC that does not contain your licensed copy of xPC Target.

See the xPC Target API Reference Guide documentation for details.

### **Can Signal Outputs from Virtual Blocks Be Tagged?**

You cannot directly tag signal outputs from virtual blocks. Instead, do the following:

- 1 Add a unity gain block (a Gain block with a gain of 1) to the model.
- 2 Connect the signal output of the virtual block to the input of the unity gain block.
- 3 Tag the output signal of the unity gain block.

### **Why Has the Stop Time Changed?**

If you change a target application step size 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

$$\text{floor}(\text{stop time}/\text{step size})$$

When you compile code for a model, Real Time Workshop 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, Real Time Workshop also 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 adjusted stop time. This might lead to fewer steps than you expect.

For example, if a model has a stop time of 2.4 and a step size of 1, Real-Time Workshop adjusts the stop time of the model to 2 at compilation. If you change the step size to .6 but do not recompile the code, the expected number of steps is 4, but the actual number of steps is 3. This is because Real-Time Workshop still uses the adjusted stop time of 2.

To avoid this problem, ensure that the original stop time (as specified in the model) is an integral multiple of the original step size.

### **Why Do I Get File System Disabled Error?**

If your target PC does not have a FAT hard disk, the monitor on the target PC displays the following error:

```
ERROR: No accessible disk found: file system disabled
```

If you do not want to access the target PC file system, you can ignore this message. If you want to access the target PC file system, add a FAT hard disk to the target PC system and reboot.

### **How can I Diagnose Network Problems with xPC Target?**

If you experience network problems when using xPC Target, 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 PC Command-Line Interface Reference

---

xPC Target provides a limited set of commands that you can use to work the target application after it has been loaded to the target PC, and to interface with the scopes for that application. This chapter is a reference of those commands.

Target PC Commands (p. 14-2)

Description of commands on the target PC for stand-alone applications that are not connected to the host PC

## Target PC Commands

The target PC command-line interface enables you to work with target and scope objects in a limited capacity. Methods let you interact directly with the scope or target. Property commands let you work with target and scope properties. Variable commands let you alias target PC command-line interface commands to names of your choice. This section provides references for the following:

- “Target Object Methods” on page 14-3
- “Target Object Property Commands” on page 14-3
- “Scope Object Methods” on page 14-5
- “Scope Object Property Commands” on page 14-8
- “Aliasing with Variable Commands” on page 14-10

Refer to Chapter 7, “Using the Target PC Command-Line Interface,” for a description of how to use these methods and commands.



## Target Object Methods

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

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

Target PC Command	Description and Syntax	MATLAB Equivalent
<code>start</code>	Start the target application currently loaded on the target PC.  Syntax: <code>start</code>	<code>tg.start</code> or <code>+tg</code>
<code>stop</code>	Stop the target application currently running on the target PC.  Syntax: <code>stop</code>	<code>tg.stop</code> or <code>-tg</code>
<code>reboot</code>	Reboot the target PC.  Syntax: <code>reboot</code>	<code>tg.reboot</code>

## Target Object Property Commands

When you are using the target PC command-line interface, target object properties are limited to parameters, signals, stop time, and sample time. Note the difference between a parameter index (0, 1, . . .) and a parameter name (P0, P1, . . .).

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

<b>Target PC Command</b>	<b>Description and Syntax</b>	<b>MATLAB Equivalent</b>
getpar	<p>Display the value of a block parameter using the parameter index.</p> <p>Syntax: getpar parameter_index</p>	get(tg, 'parameter_name')
setpar	<p>Change the value of a block parameter using the parameter index.</p> <p>Syntax: setpar parameter_index = floating_point_number</p>	set(tg, 'parameter_name', number)
stoptime	<p>Enter a new stop time. Use inf to run the target application until you manually stop it or reset the target PC.</p> <p>Syntax: stoptime = floating_point_number</p>	tg.stoptime = number
sampletime	<p>Enter a new sample time.</p> <p>Syntax: sampletime = floating_point_number</p>	tg.sampletime = number  set(tg, 'SampleTime', number)

<b>Target PC Command</b>	<b>Description and Syntax</b>	<b>MATLAB Equivalent</b>
P#	<p>Display or change the value of a block parameter. For example, P2 or P2=1.23e-4.</p> <p>Syntax: parameter_name or parameter_name = floating_point_number</p> <p>parameter_name is P0, P1, . . .</p>	<p>tg.getparam(parameter_index)</p> <p>tg.setparam(parameter_index, floating_point_number)</p>
S#	<p>Display the value of a signal. For example, S2.</p> <p>Syntax: signal_name</p> <p>signal_name is S0, S1, . . .</p>	<p>tg.getsignal(signal_index)</p>

## Scope Object Methods

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

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

column. The target object name `tg` and the scope object name `sc` are used as an example for the MATLAB methods.

<b>Target PC Command</b>	<b>Description and Syntax</b>	<b>MATLAB Equivalent</b>
<code>addscope</code>	<code>addscope scope_index</code> <code>addscope</code>	<code>tg.addscope(scope_index)</code> <code>tg.addscope</code>
<code>remscope</code>	<code>remscope scope_index</code> <code>remscope all</code>	<code>tg.remscope(scope_index)</code> <code>tg.remscope</code>
<code>startscope</code>	<code>startscope scope_index</code>	<code>sc.start</code> or <code>+sc</code>
<code>stopscope</code>	<code>stopscope scope_index</code>	<code>sc.stop</code> or <code>-sc</code>
<code>addsignal</code>	<code>addsignal scope_index =</code> <code>signal_index1,</code> <code>signal_index2, . . .</code>	<code>sc.addsignal(signal_index_vector)</code>
<code>remsignal</code>	<code>remsignal scope_index =</code> <code>signal_index1,</code> <code>signal_index2, . . .</code>	<code>sc.remsignal(signal_index_vector)</code>
<code>viewmode</code>	<p>Zoom in to one scope or zoom out to all scopes.</p> <p>Syntax: <code>viewmode scope_index</code> or left-click the scope window</p> <p><code>viewmode 'all'</code> or right-click any scope window</p> <p>Press the function key for the scope, and then press <b>V</b> to toggle <code>viewmode</code>.</p>	

<b>Target PC Command</b>	<b>Description and Syntax</b>	<b>MATLAB Equivalent</b>
ylimit	<code>ylimit scope_index</code> <code>ylimit scope_index = auto</code> <code>ylimit scope_index = num1, num2</code>	
grid	<code>grid scope_index on</code> <code>grid scope_index off</code>	

## Scope Object Property Commands

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

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

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

Target PC	MATLAB
<code>numsamples scope_index = number</code>	<code>sc.NumSamples = number</code>
<code>decimation scope_index = number</code>	<code>sc.Decimation = number</code>
<code>scopemode scope_index = 0 or numerical, 1 or redraw, 2 or sliding, 3 or rolling</code>	<code>sc.Mode = 'numerical', 'redraw', 'sliding', 'rolling'</code>
<code>triggermode scope_index = 0, freerun, 1, software, 2, signal, 3, scope</code>	<code>sc.TriggerMode = 'freerun', 'software', 'signal', 'scope'</code>
<code>numprepostsamples scope_index = number</code>	<code>sc.NumPrePostSamples = number</code>
<code>triggersignal scope_index = signal_index</code>	<code>sc.TriggerSignal = signal_index</code>
<code>triggersample scope_index = number</code>	<code>sc.TriggerSample = number</code>
<code>triggerlevel scope_index = number</code>	<code>sc.TriggerLevel = number</code>

<b>Target PC</b>	<b>MATLAB</b>
<code>triggerslope scope_index = 0, either, 1, rising, 2, falling</code>	<code>sc.TriggerSlope = 'Either', 'Rising', 'Falling'</code>
<code>triggerscope scope_index2 = scope_index1</code>	<code>sc.TriggerScope = scope_index1</code>
<code>triggerscopesample scope_index= integer</code>	<code>sc.TriggerSample = integer</code>
Press the function key for the scope, and then press <b>S</b> or move mouse into the scope window.	<code>sc.trigger</code>

## Aliasing with Variable Commands

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

Target PC Command	Description and Syntax	MATLAB Equivalent
setvar	<p>Set a variable to a value. Later you can use that variable to do a macro expansion.</p> <p>Syntax: setvar variable_name = target_pc_command</p> <p>For example, you can type setvar aa=startscope 2, setvar bb=stopscope 2.</p>	None
getvar	<p>Display the value of a variable.</p> <p>Syntax: getvar variable_name</p>	None
delvar	<p>Delete a variable.</p> <p>Syntax: delvar variable_name</p>	None
delallvar	<p>Delete all variables.</p> <p>Syntax: delallvar</p>	None
showvar	<p>Display a list of variables.</p> <p>Syntax: showvar</p>	None



# Obsoleted xPC Target User Interfaces

---

This chapter describes the obsoleted xPC Target tools, `xpcsetup` and `xpcrctool`. However, their presence is not guaranteed for future releases. Use the xPC Target Explorer, `xpcexplr`, instead. xPC Target Explorer consolidates the functionality of both `xpcsetup` and `xpcrctool` into one comprehensive GUI.

xPC Setup (p. 15-2)

How to use the xPC Target setup tool, `xpcsetup`

Control with xPC Target Remote  
Control Tool (p. 15-12)

How to use the xPC Remote Control tool, `xpcrctool`

## xPC Setup

The following procedures describe how to use the `xpcsetup` tool to configure your xPC Target environment. In the future, this tool will be obsolete. You should use xPC Target Explorer, `xpcexplr`, instead.

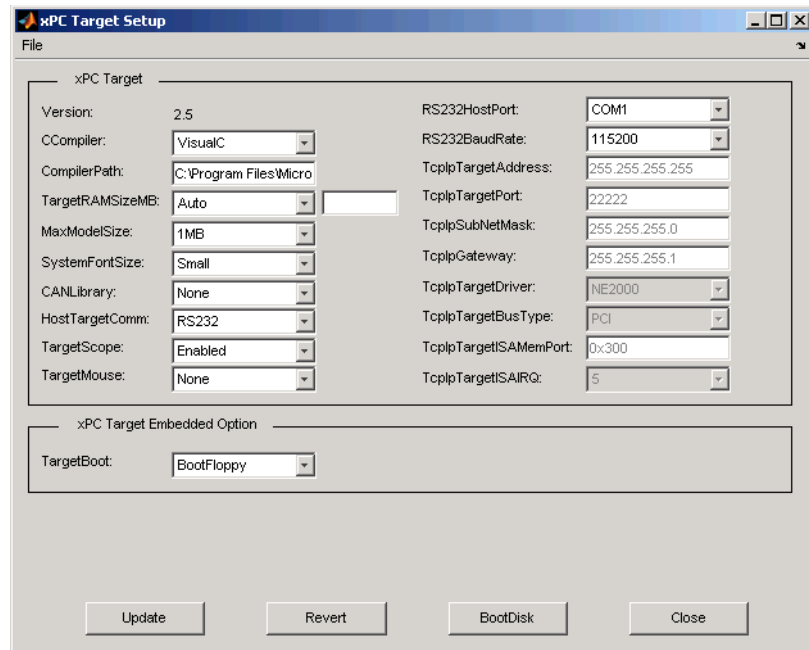
### Environment Properties for Serial Communication

The xPC Target environment is defined by a group of properties. These properties give xPC Target information about the software and hardware products that it works with. You might change some of these properties often, while others you will rarely want to change.

After you have installed xPC Target, you can specify the environment properties for the host and target computers. Note that you must specify these properties before you can build and download a target application:

- 1 In the MATLAB Command Window, type  
`xpcsetup`

The **xPC Target Setup** window opens.



The **xPC Target Setup** window has two sections:

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

If your license does not include the embedded option, the **TargetBoot** list is disabled (grayed out) with **BootFloppy** as your only choice. With the **xPC Target Embedded Option** installed, you have the additional choices of **DOSLoader** and **StandAlone**.

- 2 From the **CCompiler** list, select either **VisualC** or **Watcom**.
- 3 In the **CompilerPath** box, enter the root path where you installed your C/C++ compiler.
- 4 From the **HostTargetComm** list, select **RS-232**.

- 5** From the **RS232HostPort** list, select either COM1 or COM2 for the connection on the host PC. xPC Target determines the COM port you use on the target computer automatically.
- 6** From the **RS232BaudRate** list, select the baud rate for your serial connection.
- 7** When you finish changing the properties, click the **Update** button.

xPC Target updates the environment with the new properties.

You do not have to exit and restart MATLAB after making changes to the xPC Target environment, even if you change the communication between the host and target from RS-232 to TCP/IP. However, you do have to recreate the target boot disk and rebuild the target application from the Simulink model.

## Environment Properties for Network Communication

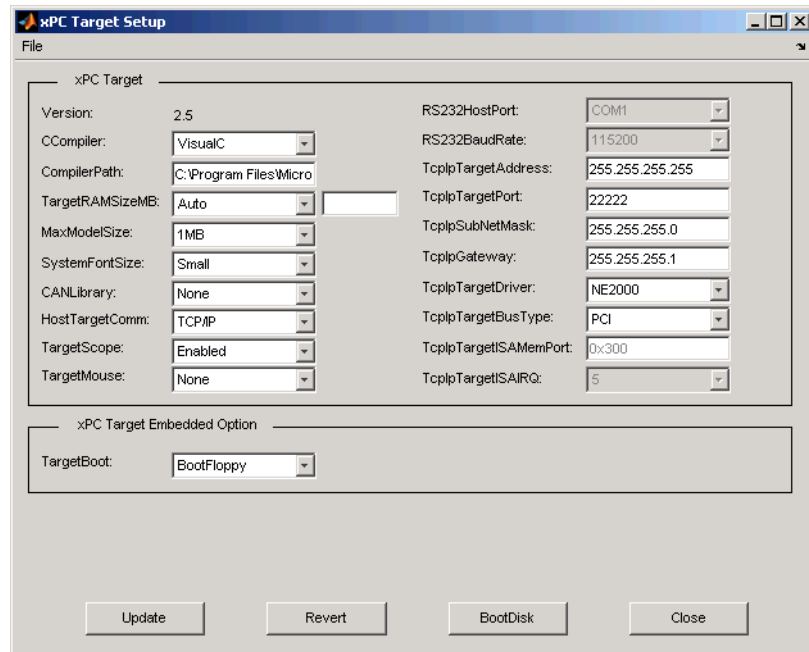
The xPC Target environment is defined by a group of properties. These properties give xPC Target information about the software and hardware that it works with. You might change some of these properties often, while others you will rarely want to change.

After you have installed xPC Target, you can specify the environment properties for the host and target computers. Note that you must specify these properties before you can build and download a target application

- 1** In the MATLAB window, type

```
xpcsetup
```

The **xPC Target Setup** window opens.



The **xPC Target Setup** window has two sections:

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

If your license does not include the embedded option, the **TargetBoot** list is disabled (grayed out) with **Boot Floppy** as your only choice. With the xPC Target Embedded Option installed, you have the additional choices of **DOSLoader** and **StandAlone**.

- 2 From the **CCompiler** list, select either **VisualC** or **Watcom**.
- 3 In the **CompilerPath** box, enter the root path where you installed your C/C++ compiler.

**4** From the **HostTargetComm** list, select TCP/IP.

The TCP/IP text boxes become active.

You must enter the following properties with the correct values according to your LAN environment. Ask your system administrator for values for the following settings:

- **TcpIpTargetAddress** — This is the IP address for your target PC. An example of an IP address is 192.168.0.1.
- **TcpIpSubNetMask** — This is the Subnet Mask address of your LAN. An example of a Subnet Mask address is 255.255.0.0.

You enter the following properties depending on your specific circumstances:

- **TcpIpTargetPort** — This property is set by default to 22222. This value should not cause any problems, because this number is higher than the reserved area (telnet, ftp, ...) and it is only of relevance on the target PC. If necessary, you can change this property value to any value higher than 20000 and less than 65536.
- **TcpIpGateway** — This property is set by default to 255.255.255.255. This means that you do not use a gateway to connect to your target PC. If you communicate with the target PC from within your LAN, you might not need to define a gateway and change this setting.

If you communicate from a host PC located in a LAN different from your target PC, you need to define a gateway and enter its IP address. This is especially true if you want to work over the Internet. Ask your system administrator for the IP address of the appropriate gateway.

The following properties are specific for the Ethernet card on your target PC:

- **TcpIpTargetDriver** — From the list, select NE2000, SMC91C9X, I82559, or RTLANCE. This property is set by default to NE2000.
- **TcpIpTargetBusType** — This property is set by default to PCI. If `TcpIpTargetBusType` is set to PCI, then the properties `TcpIpISAMemPort` and `TcpIpISAIRQ` have no effect on TCP/IP communication and are disabled (grayed out). If you are using an ISA bus Ethernet card, set `TcpIpTargetBusType` to ISA and enter values for `TcpIpISAMemPort` and `TcpIpISAIRQ`.

- **TcpIpISAMemPort** and **TcpIpISAIRQ** — 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.
- 5 When you finish changing the properties, click the **Update** button.  
xPC Target updates the environment with the new properties.

## Creating an xPC Target Boot Disk

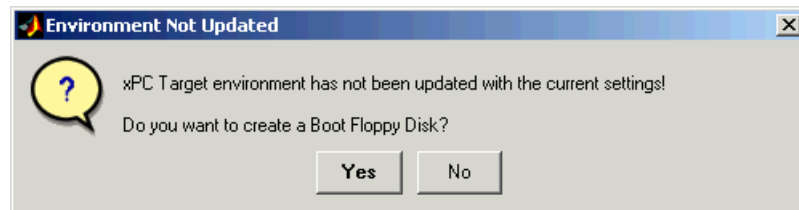
- 1 In the MATLAB Command Window, type  

```
xpcsetup
```

The **xPC Target Setup** window opens.

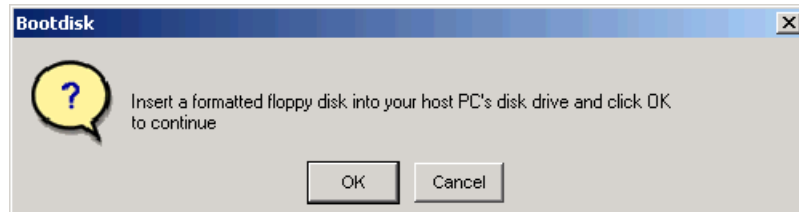
- 2 Click the **BootDisk** button.

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



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

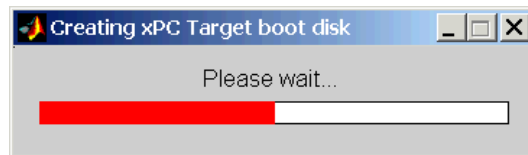
After you update the current properties and click the **BootDisk** button, the following message box opens.



- 3 Insert a formatted 3.5 inch floppy disk into the host PC disk drive, and then click **OK**.

All data on the disk will be erased.

xPC Target displays the following dialog box while creating the boot disk. The process takes about 1 to 2 minutes.



- 4 Close the **xPC Target Setup** window.
- 5 Remove the target boot disk from the host PC disk drive and insert it into your target PC disk drive.

Your next task is to install the software on the target PC and test your installation.

## Changing Environment Properties with a Graphical Interface

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

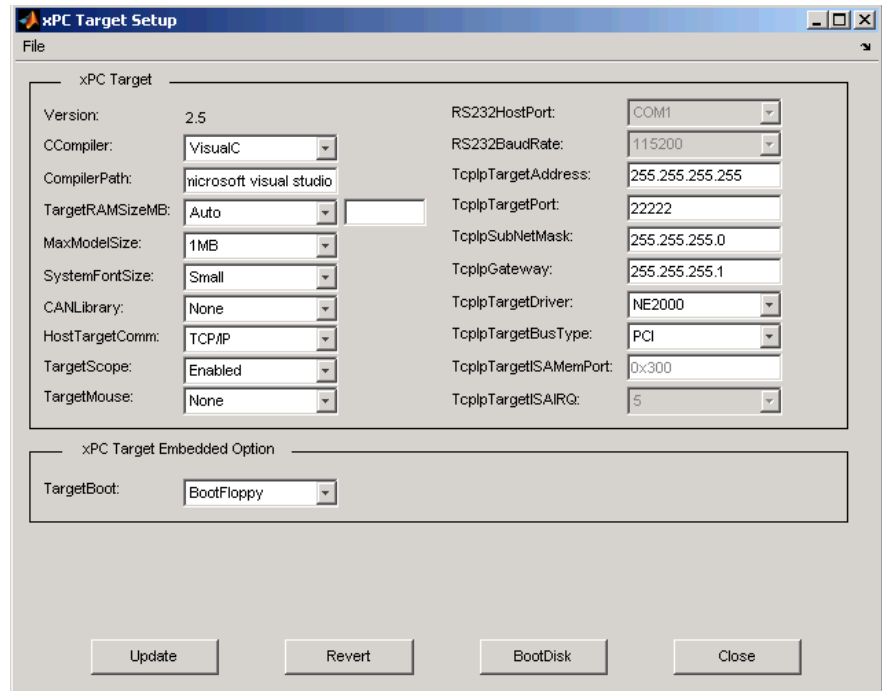
To change an environment property using the xPC Target GUI, use the following procedure:



1 In the MATLAB window, type

```
xpcsetup
```

MATLAB opens the **xPC Target Setup** window.



The **xPC Target Setup** window has two sections: **xPC Target** and **xPC Target Embedded Option**.

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

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

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

- 3 Click the **Update** button.

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

## Saving and Loading the Environment Properties

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

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

The **Save xPC Target Environment** dialog box opens.

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

xPC Target saves the current environment properties.

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

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

The **Load xPC Target Environment** dialog box opens.

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

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

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

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

## Control with xPC Target Remote Control Tool

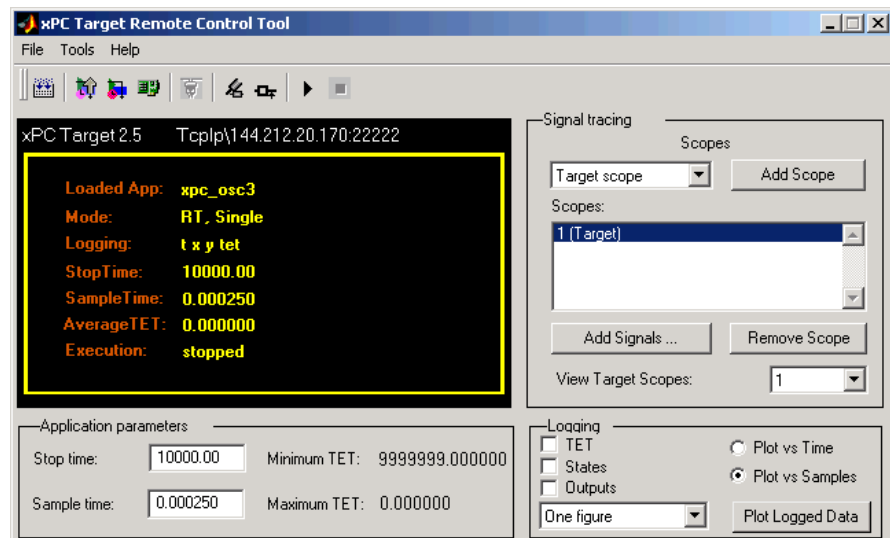
The following procedures describe how to use the `xpcrcctool` tool to control your xPC Target environment. In the future, this tool will be obsolete. You should use xPC Target Explorer, `xpcexplr`, instead.

This procedure assumes you have created an xPC Target boot disk and you have booted the target PC. While the xPC Target Remote Control Tool has the ability to build and download a target application, this procedure begins with a target application already downloaded to the target PC.

- 1 In the MATLAB window, type

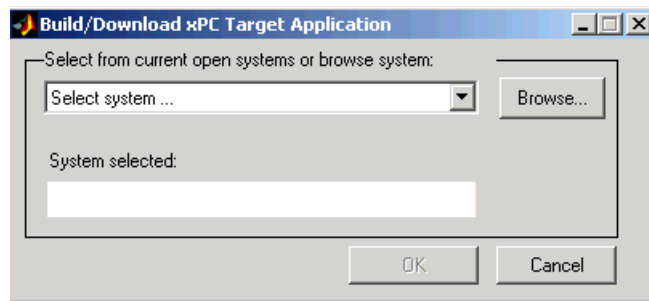
```
xpcrcctool
```

The **xPC Target Remote Control Tool** window opens, connects to the target PC, and displays information about the previously loaded target application.



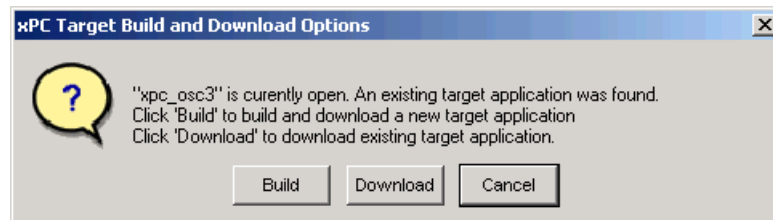
- 2 If you want to rebuild the target application, from the **File** menu, click **Build/Load Application**.


The **Build/Download xPC Target Application** dialog box opens.




- 3 From the current open systems list, select the name of a Simulink model (.mdl) you have open, or click the **Browse** button and select the name of a Simulink model or a previously built target application (.dlm). Click **OK**.

The **Build and Download Options** dialog box opens.



- 4 Select a download option.
  - If you click **Build**, Real-Time Workshop creates a target application from the Simulink model and downloads it to the target PC.
  - If you click **Download**, xpcrc tool looks for a previously built target application for the selected Simulink model and downloads it to the target PC.
- 5 From the toolbar, click the Start target application button . The target application begins running on the target PC, and stops when it reaches the stop time.
- 6 In the **Stop time** box, enter a new stop time. For example, enter  
inf

- 7 Again, click the Start target application button. The target application now runs until you stop it.
- 8 From the toolbar, click the Stop target application button .

## Signal Tracing with xPC Target Remote Control Tool

Opening an xPC Target scope window on the host or target PC allows you to view signals while running and testing your target application.

After you create and download a target application to the target PC, you can view signals. This procedure uses a Simulink model `xpc_osc4.mdl` as an example, and assumes you have created a target application and downloaded it to the target PC. See “xPC Target Application” in Chapter 3 of the xPC Target Getting Started documentation:

- 1 In the MATLAB window, type

```
xpcrcctool
```

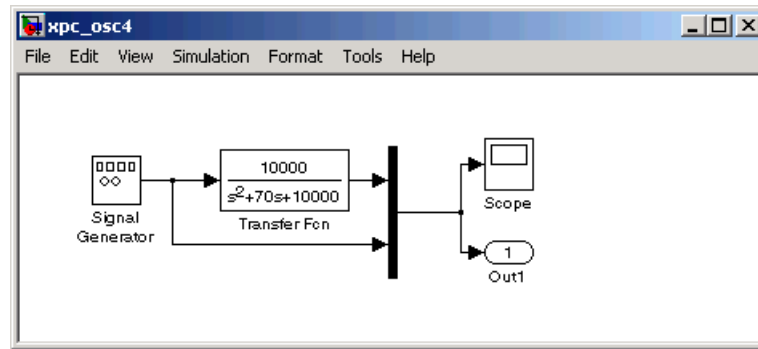
The **xPC Target Remote Control Tool** window opens and connects to the target application.

- 2 From the scope type list, select either Target scope, Host scope, or File scope. For example, select Target scope, and then click **Add Scope**.

xPC Target creates a scope on the target PC, and in the **Scopes** list, displays 1 (Target).

- 3 Click **Add Signals**.

The **xPC Target Simulink Viewer** window opens with a diagram of your Simulink model.



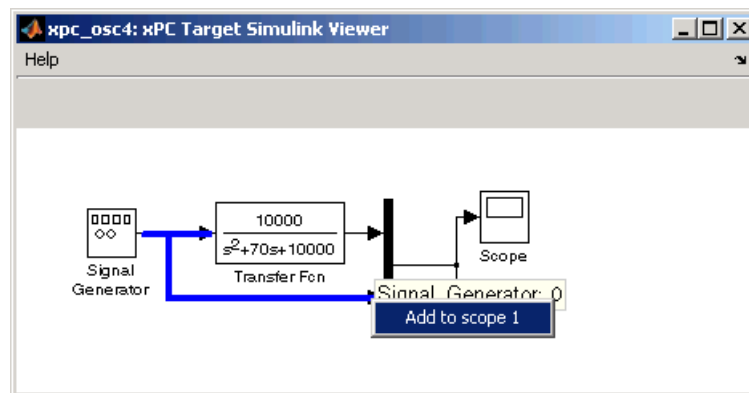
**4** Move the mouse pointer over the signal lines.

- A blue line indicates that you can add this signal to a scope.
- A red line, with the message Signal cannot be monitored, indicates that you cannot add this signal to a scope.

Note that you can add a signal while the scope is running or stopped. You do not need to stop the scope to add or remove signals.

**5** Point to a signal line and right-click. For example, right-click over the signal from the Signal Generator.


A menu opens with a list of scopes and options to add a signal or to remove a previously added signal.



- 6 From the menu, click an option. For example, click **Add to scope 1**.

The signal from the Signal Generator is added to the scope on the target PC screen.

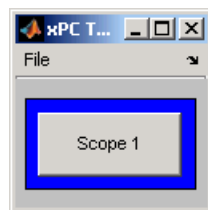
- 7 Point to the signal from the Transfer Function block, right-click, and select **Add to scope 1**.

- 8 From the toolbar, click the start button . The target application and scope begin running on the target PC, and then stop when the target application reaches the stop time.

### Setting Trigger for Signals with the Target Scope Manager

- 1 In the **xPC Target Remote Control Tool** window, from the **Tools** menu, click **Target Scope Manager**.

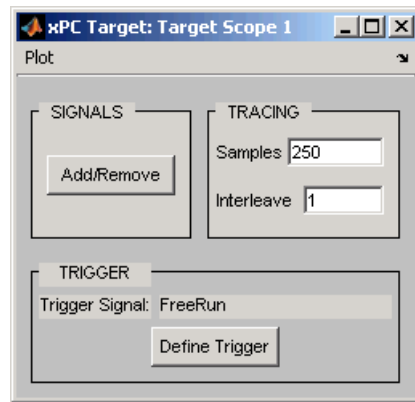
The scope manager window opens. This window is the root window of the xPC Target scope graphical interface, and it displays the scopes that you have added.




- 2 Point to the **Scope 1** button and right-click. A menu opens.
- 3 From the menu, click **Properties**.

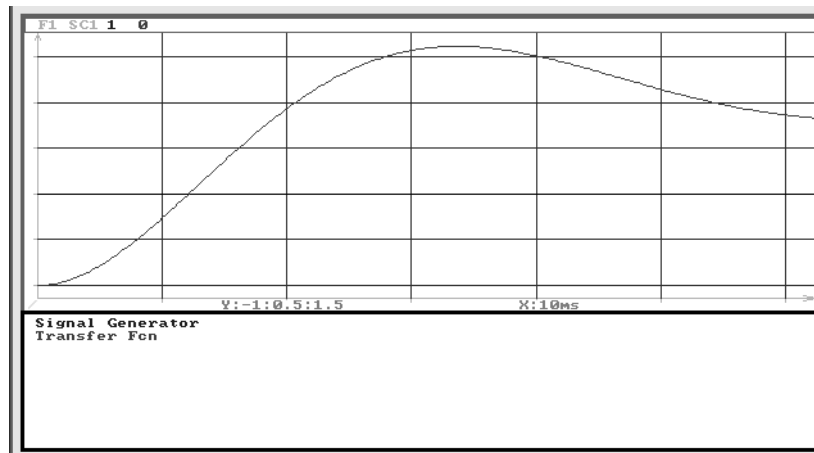
The **xPC Target: Target Scope 1** window opens.





- 4 Click **Define Trigger**. The **xPC Target: Trigger for Target Scope 1** window opens.
- 5 From the **Mode** list, select Software, Signal, or Scope. For example, select Signal.
- 6 From the **Slope** list, select EITHER, RISING, or FALLING. For example, select RISING.
- 7 From the **Signal** list, click a signal to trigger the scope. For example, click Signal\_generator.
- 8 In the scope manager window opens, right-click the **Scope 1** button and select Start.
- 9 In the **xPC Target Remote Control Tool** window, from the toolbar, click the start button .

The target application and scope begin to run on the target PC, and then stop when the target application reaches the stop time.



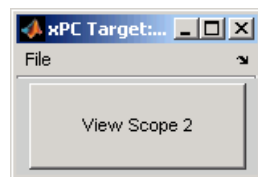
### Adding and Removing Signals with the Host Scope Manager

- 1 In the **xPC Target Remote Control Tool** window, from the scope type list, select **Host** scope, then click **Add Scope**.

xPC Target creates a scope on the host PC, and in the **Scopes** list, displays 2 (Host).

- 2 In the **xPC Target Remote Control Tool** window, from the **Tools** menu, click **Host Scope Manager**.

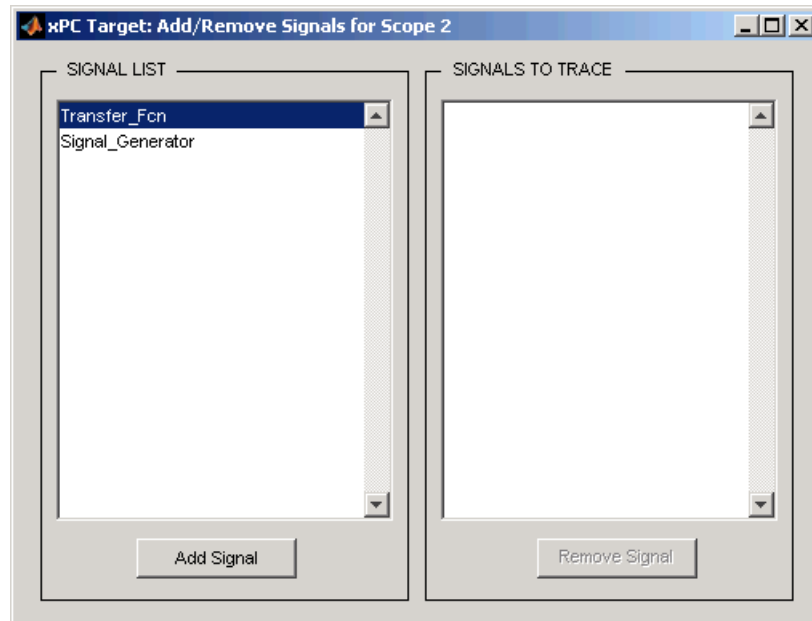
The scope manager window for scopes of type host opens.



- 3 Click the button for a scope. For example, click **View Scope 2**. The **xPC Target: Scope 2** window opens.

- 4 In the scope window, click the **Add/Remove** button.

The **Add/Remove Signals for Scope 2** dialog box opens. It allows you to specify the signals to trace.

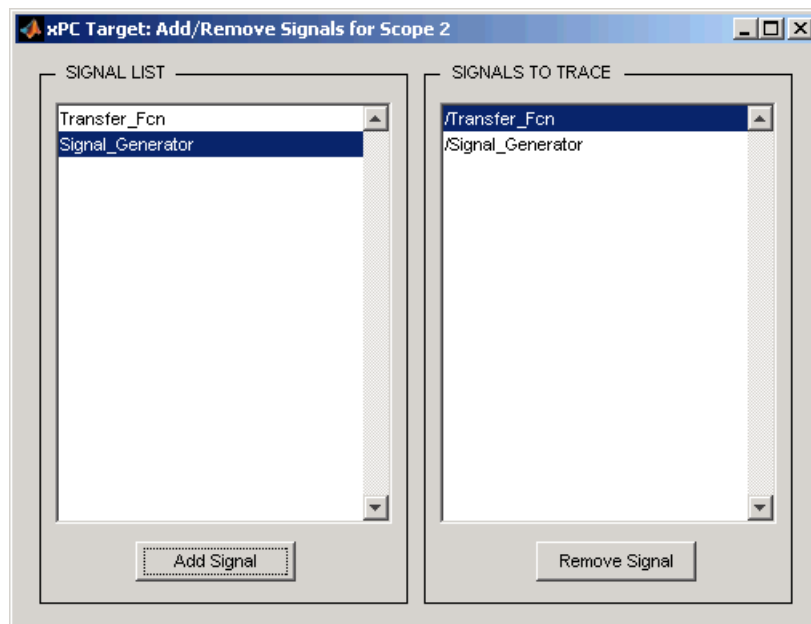


The **SIGNAL** list box displays all the available signals from the target application. The names of the signals shown correspond to the block names within the Simulink model `xpc_osc4.mdl`. The block name indicates the output signal from that block.


Click a block name to highlight it, and then click the **Add Signal** button to move the signal to the **SIGNALS TO** box on the right of the window. The **SIGNALS TO** box contains the signals to be traced by this scope.

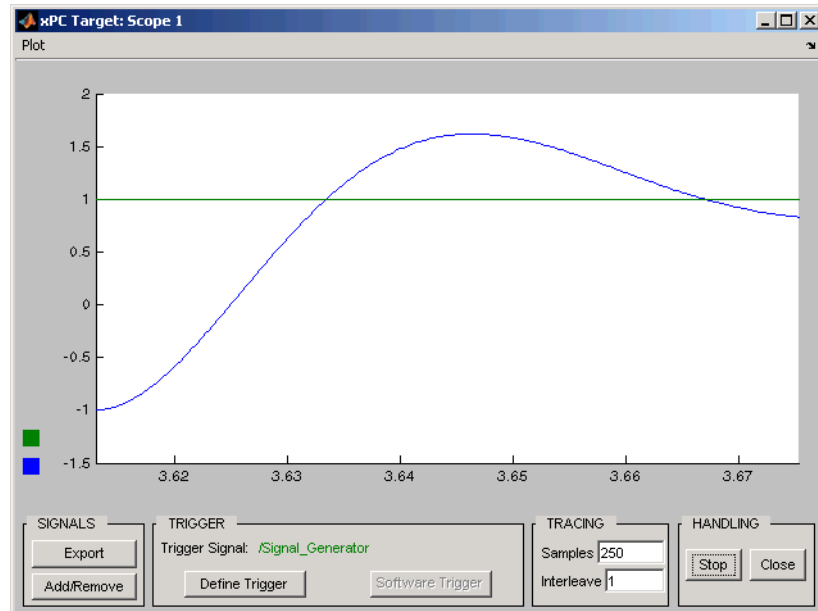
- 5 From the **SIGNAL** list box, click `Transfer_Fcn`, and then click the **Add Signal** button. Similarly, add the `Signal_Generator` signal.

Changes to the **Add/Remove Signals for Scope 2** dialog box are shown below. You can remove the signals to trace by clicking the block name in the **SIGNALS TO** box, and then clicking the **Remove Signal** button.



During the next steps, you can leave the **Add/Remove Signals for Scope 2** dialog box open, or close and reopen it without restrictions.

- 6 In the scope window, click the **Start** button at the bottom right of the window.
- 7 In the **xPC Target Remote Control Tool** window, from the toolbar, click the start button . The target application and scope begin to run on the target PC, and then stop when they reach the stop time.



If you are using a scope of type host, there is a delay between collecting data packages because of the communication overhead between the target PC and the host PC. If you are using a scope of type target, the scope window is updated faster than when using a scope on the host PC.

### Saving Data from a Scope of Type Host to the MATLAB Workspace

While a target application is running, data is saved in an xPC Target Scope buffer. You can export the buffer to MATLAB. Exporting data with the **xPC Target Scope** window does not require you to add output blocks to your Simulink model, nor does it require you to activate the logging of signals. You can select the signals to collect, and you can capture unexpected outputs during a run.

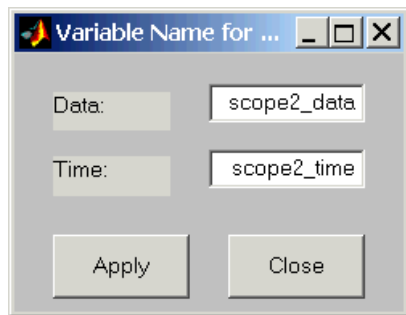
After you complete a run with a scope, you can move data from the last data package collected to the MATLAB workspace:

- 1 In the **xPC Target Remote Control Tool** window, from the **Tools** menu, click **Host Scope Manager**.

The scope manager window and the **xPC Target Scope** window open.

- 2 In the **xPC Target Scope** window, from the **Plot** menu, click **Variable Name for Export**.

The **Variable Name for Export** dialog box opens.



- 3 In the **Data** and **Time** text boxes, enter the names of the MATLAB variables to save data from a trace. Click the **Apply** button, and then click the **Close** button. The default name for the time vector is `scopen_time`, and the default name for the signal vector is `scopen_data` where *n* is the scope number.
- 4 In the **xPC Target Scope** window, click the **Export** button. You can export data regardless of whether a scope is started or stopped.

**5** In the MATLAB window, type

```
whos
```

MATLAB displays a list of variables and their descriptions. For example,

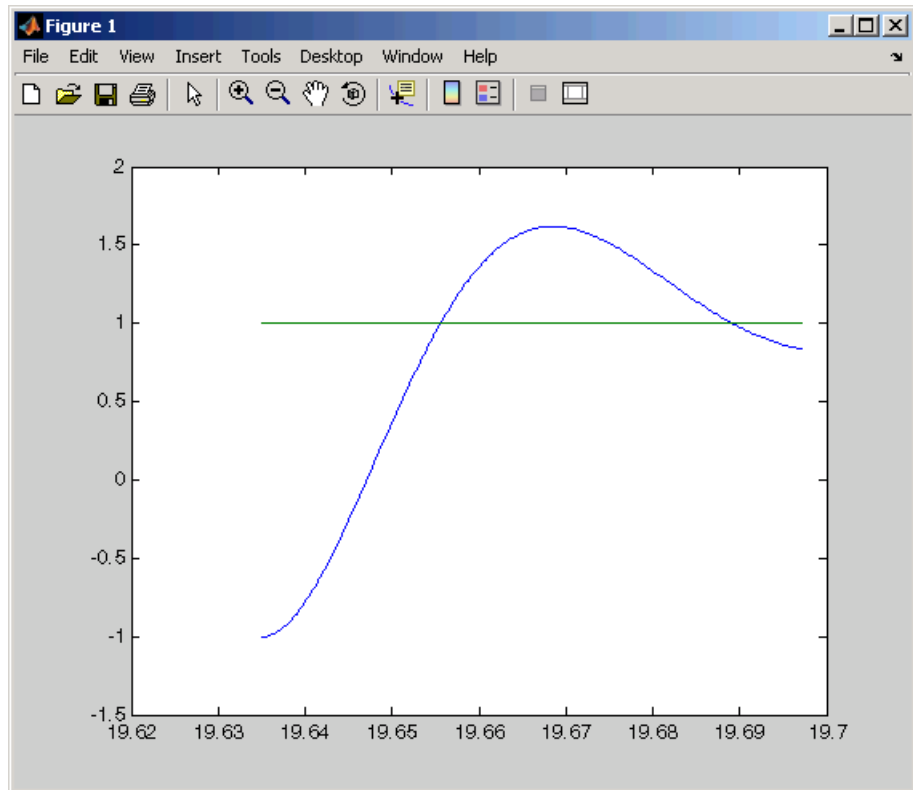
```
Name           Size      Bytes  Class
scope2_data    250x2     4000   double array
scope2_time    1x250     2000   double array
tg             1x1                xpctarget.xpc
Grand total is 752 elements using 6000 bytes
```

You can now save or further analyze the data using the MATLAB variables.

**6** Type

```
plot(scope2_time, scope2_data)
```

MATLAB plots the variables `scope1_time` and `scope1_data` in a new window.



- 7 Close the scope manager window by using one of the following procedures:
  - From the **File** menu, click **Close All Scopes**.
  - From the **File** menu, click **Close Scope Manager**.
- 8 A message box opens asking whether you want to save the current scope state. Use one of the following procedures:
  - If you do not want to save the scope state, click **No**.
  - If you want to save the scope state, click **Yes**. The **Save Scopes** dialog box opens. Enter the name of a file, and then click **Save**.



## Signal Logging with xPC Target Remote Control Tool

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 the model `xpc_osc4.mdl` as an example and assumes you have created a target application and downloaded it to the target PC. See “xPC Target Application” in Chapter 3 in the xPC Target Getting Started documentation.

---

**Note** To use the xPC Target Remote Control Tool for signal logging, you need to add an Output block to your Simulink model, and you need to activate logging on the **Data Import/Export** pane in the **Configuration Parameters** dialog.

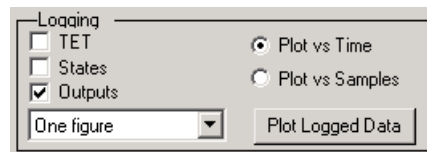
---


- 1 In the MATLAB window, type

```
xpcrctool
```

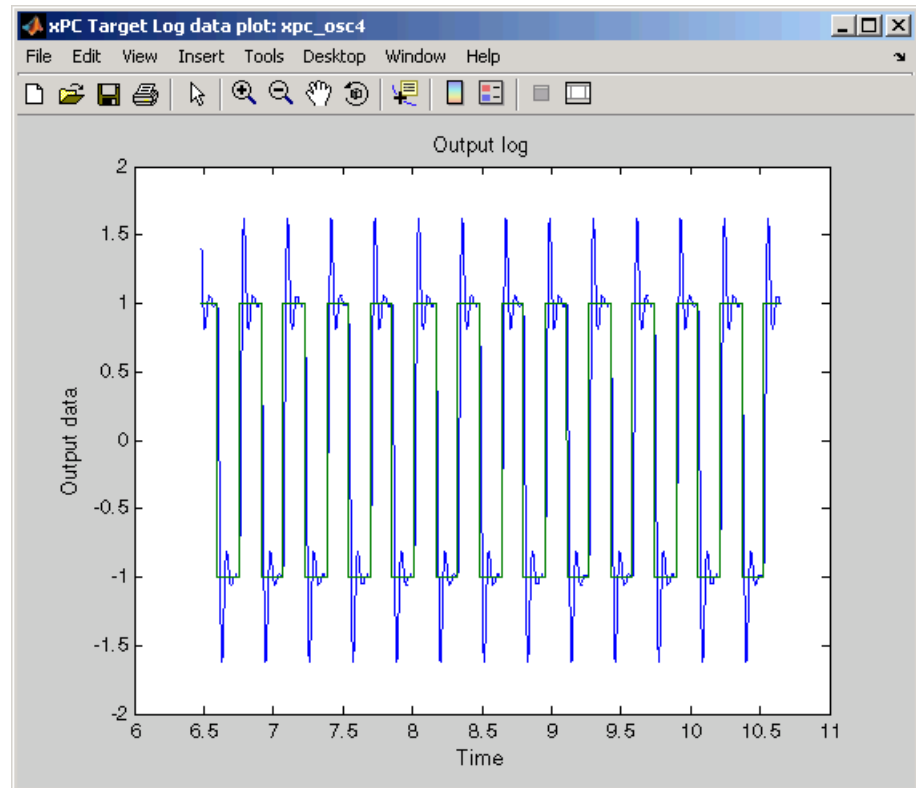
The **xPC Target Remote Control Tool** window opens, connects to the target PC, and displays information about a previously loaded target application.

- 2 Select the **Outputs** check box and the **Plot vs Time** option.



- 3 From the toolbar, click the start button . The target application begins running on the target PC, and then stops when it reaches the stop time.
- 4 Click the **Plot Logged Data** button.

MATLAB opens a figure window and draws a plot of the outputs.



---

**Note** Logged data is available only when the target application is not running.

---

## Parameter Tuning with xPC Target Remote Control Tool

You can change parameters in your target application using the xPC Target Simulink Viewer whether the target application is stopped or running.

After you create and download a target application to the target PC, you can view signals. This procedure uses the Simulink model `xpc_osc4.mdl` as an

example, and assumes that you have created a target application and downloaded it to the target PC. See “xPC Target Application” in Chapter 3 of the xPC Target Getting Started documentation:

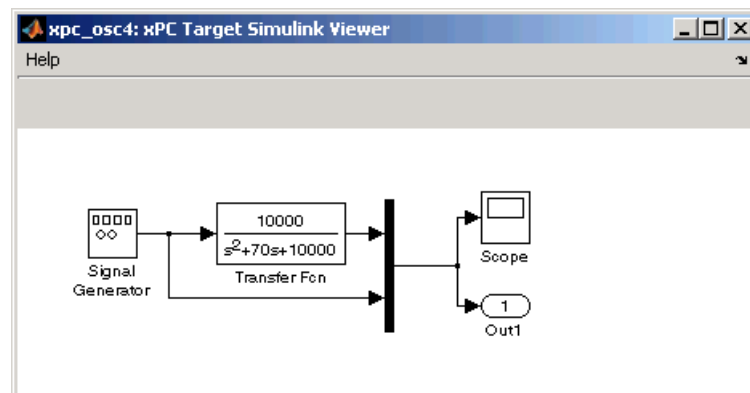
- 1 In the MATLAB window, type

```
xpcrc tool
```

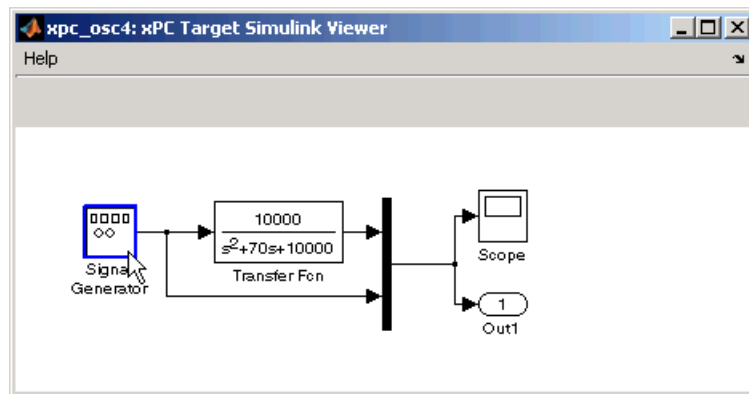
The **xPC Target Remote Control Tool** window opens and connects to the target application.

- 2 From the **Tools** menu, click **Tune Parameters**.

The **xPC Target Simulink Viewer** window opens with a diagram of your Simulink model.



- 3 Move the mouse pointer over the blocks.
  - A blue highlight around a block indicates that this block contains tunable parameters.
  - A red highlight around a block indicates that there are no tunable parameters.



- 4 Point to a block and right-click. For example, right-click over the Signal Generator block.

The **Block Parameters** dialog box opens.

- 5 Enter new parameters, and then click **OK**.

The new parameter values are downloaded to the target application. If the target application is running, you can immediately see the effect of the change.

# Function Reference

---

## Functions – Categorical List

This topic contains reference pages for xPC Target functions.

“Software Environment” on page 16-2	Help you define the software and hardware environment of the host PC as well as the target PC
“GUI” on page 16-3	Open xPC Target ancillary GUIs
“Test” on page 16-3	Run tests from the MATLAB Command Window
“Target Objects” on page 16-3	Control a target application on the target PC from the host PC
“Scope Objects” on page 16-6	Control scopes on your target PC
“File and File System Objects” on page 16-7	Control file and file system objects in the target PC file system

## Software Environment

<code>getxpcenv</code>	List environment properties in the MATLAB window or assign the list as a cell array to a MATLAB variable.
<code>setxpcenv</code>	First of two steps to change environment properties. See also <code>updatexpcenv</code> .
<code>updatexpcenv</code>	Change the current environment properties to equal the new properties entered using the function <code>setxpcenv</code> .
<code>xpcbootdisk</code>	Create a boot disk containing the kernel according to the current environment properties.
<code>xpcwwwenable</code>	Disconnect the target PC from the current client application

## GUI

<code>xpcexplr</code>	Open the xPC Target Explorer window.
<code>xpcrcctool</code>	Open the remote control tool on the host PC for running the target application on the target PC.
<code>xpcscope</code>	Open the <b>xPC Target:Host Manager</b> scope manager window on the host PC for scopes with type host.
<code>xpcsetup</code>	Open the xPC Target setup window.
<code>xpctargetspy</code>	Open the <b>Real-Time xPC Target Spy</b> window on the host PC. Use this GUI to upload the target PC screen to the host PC.
<code>xpctgscope</code>	Open the <b>xPC Target:Target Manager</b> scope manager window on the host PC for scopes with type target.

## Test

<code>getxpcpci</code>	Determine which PCI boards are installed in the target PC.
<code>xpctargetping</code>	Test the communication between the host PC and the target PC.
<code>xpctest</code>	Test the xPC Target installation.

## Target Objects

The target object methods allow you to control a target application on the target PC from the host PC. You enter target object methods in the MATLAB window on the host PC or use M-file scripts. To access the M-file help for these methods, use the syntax

```
help xpctarget.xpc/method_name
```

If you want to control the target application from the target PC, use target PC commands. See “Using the Target PC Command-Line Interface” on page 7-1.

<code>xpc</code>	Call the target object constructor, <code>xpctarget.xpc</code>
<code>xpctarget.xpc</code>	Create a target object on the host PC (constructor).
<code>delete</code>	Remove a target object on the host PC (destructor).
<code>set (target object)</code>	Set writable target object properties to the specified values.
<code>get (target object)</code>	Return the values of readable properties from a target object.
<code>start (target object)</code>	Start the execution of a target application on the target PC.
<code>stop (target object)</code>	Stop the execution of a target application on the target PC.
<code>load</code>	Download a target application from the host PC to the target PC.
<code>unload</code>	Unload a target application from the target PC. If a target application is running, it is stopped and unloaded.
<code>addscope</code>	Create a new scope with type 'host', 'target', or 'file' on the target PC.
<code>setparam</code>	Set writable target object parameters to the specified values.
<code>getparam</code>	Return the value of the specified parameter index from the target object.
<code>getparamid</code>	Get a parameter index or property name from the parameter list
<code>getparamname</code>	Get the block path and parameter name from the index list



<code>getscope</code>	Return the properties of a previously created scope from the target PC. The scope properties can be assigned to a MATLAB variable to create a scope object.
<code>getsignal</code>	Return the value of the specified signal index from the target object.
<code>getsignalid</code>	Get the signal index or signal property from the signal list
<code>getsignalname</code>	Get the signal name from the index list
<code>remscope</code>	Remove a scope from the target PC. This method does not remove the scope object on the host PC that represents the scope.
<code>getparamid</code>	Return the property name or index of a parameter from the target object.
<code>getsignalid</code>	Return the property name or index of a signal from the target object.
<code>saveparamset</code>	Save the parameter values of the current target application.
<code>loadparamset</code>	Restore the parameter values saved in the specified file.
<code>getlog</code>	Upload and return one of the data logs from the target PC to the host PC (TimeLog, StateLog, OutputLog, TETLog).
<code>reboot</code>	Reboot the target PC. If a target application is running, the target application is stopped, and then the target PC is rebooted.
<code>close</code>	Close the serial connection to the target PC so that the host PC can use the COMM port for another device.
<code>targetping</code>	Test communication between a host and its target computers.

## Scope Objects

The scope object methods allow you to control scopes on your target PC.

If you want to control the target application from the target PC, use target PC commands. See Chapter 7, “Using the Target PC Command-Line Interface.”

<code>set (scope object)</code>	Set writable scope object properties to the specified values. For a list of writable values, use <code>set(scope_object)</code> .
<code>get (scope object)</code>	Return the values of readable properties from a scope object.
<code>addsignal</code>	Add a signal to a scope and a scope object. Specify the signal using the signal index from the target object.
<code>remsignal</code>	Remove a signal from a scope and a scope object. Specify the signal using signal index from the scope object.
<code>start (scope object)</code>	Start a scope, but do not necessarily start the acquisition of data. The acquisition of data depends on the trigger mode.
<code>stop (scope object)</code>	Stop a scope and the acquisition of data.
<code>trigger</code>	Start the acquisition of data from the target application using a scope: <ul style="list-style-type: none"><li>• If <code>TriggerMode = 'Software'</code>, then the scope can only be triggered by software using the method <code>sc.trigger</code>.</li><li>• If <code>TriggerMode = 'FreeRun', 'Signal',</code> or <code>'Scope'</code>, you can trigger the scope by software before the scope is triggered by one of the other modes.</li></ul>

## File and File System Objects

xPC Target uses the `xpctarget.ftp` object to represent a target PC file, such as a signal data file created by an xPC Target scope of type `file`. Use this object to perform file transfer operations on that file. xPC Target uses the `xpctarget.fs` object to represent the target PC file system. Use this object to perform file system manipulations.

Both `xpctarget.ftp` and `xpctarget.fs` belong to the `xpctarget.fsbase` object. This object encompasses the methods common to `xpctarget.ftp` and `xpctarget.fs`. xPC Target creates the `xpctarget.fs` base object when you create either an `xpctarget.ftp` or `xpctarget.fs` object.

This section includes the following topics:

- “`xpctarget.fsbase` Functions” on page 16-7 — List of methods with a brief description
- “`xpctarget.ftp` Functions” on page 16-8 — List of methods with a brief description
- “`xpctarget.fs` Functions” on page 16-8 — List of methods with a brief description

It concludes with a more complete description of these methods.

Refer to Chapter 8, “Working with Target PC Files and File Systems,” for a description of how to use these objects and methods.

### `xpctarget.fsbase` Functions

You enter `xpctarget.fsbase` object methods in the MATLAB Command Window on the host PC or use M-file scripts.

You can call the `xpctarget.fsbase` methods for both `xpctarget.ftp` and `xpctarget.fs` objects.

<code>cd</code>	Change directory on target PC.
<code>dir</code>	List the contents of the current directory on the target PC.
<code>mkdir</code>	Make a directory on the target PC.

<code>pwd</code>	Retrieve the current directory path of the target PC.
<code>rmdir</code>	Remove the specified directory from the target PC.

### **xpctarget.ftp Functions**

The `xpctarget.ftp` methods allow you to work with a target PC file, such as a signal data file, from the host PC. You enter target object methods in the MATLAB window on the host PC or use M-file scripts.

<code>get (ftp)</code>	Retrieve a copy of the requested file from the target PC.
<code>put</code>	Copy a file from the host PC to the target PC.

### **xpctarget.fs Functions**

The `xpctarget.fs` methods allow you to work with the target PC file system from the host PC. You enter target object methods in the MATLAB window on the host PC or use M-file scripts.

The `xpctarget.fs` methods are listed in the following table.

<code>diskinfo</code>	Get information about a target PC drive.
<code>fclose</code>	Close one or more open target PC files (similar to MATLAB <code>fclose</code> function).
<code>fileinfo</code>	Get target PC file information.
<code>filetable</code>	Get information about open files in the target PC file system.
<code>fopen</code>	Open a target PC file for reading (similar to MATLAB <code>fopen</code> function).
<code>fread</code>	Read an open target PC file (similar to MATLAB <code>fread</code> function).
<code>fwrite</code>	Write binary data to an open target PC file (similar to MATLAB <code>fwrite</code> function).

<code>getfilesize</code>	Get the size (in bytes) of a file on the target PC.
<code>removefile</code>	Remove a file from the target PC.

## **Functions – Alphabetical List**

This section contains function reference pages listed alphabetically.

**Purpose** Create one or more scopes on the target PC

**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 PC command line** — When you are using this command on the target PC, you can only add a scope of type target.

```
addscope
addscope scope_number
```

<b>Arguments</b>	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.

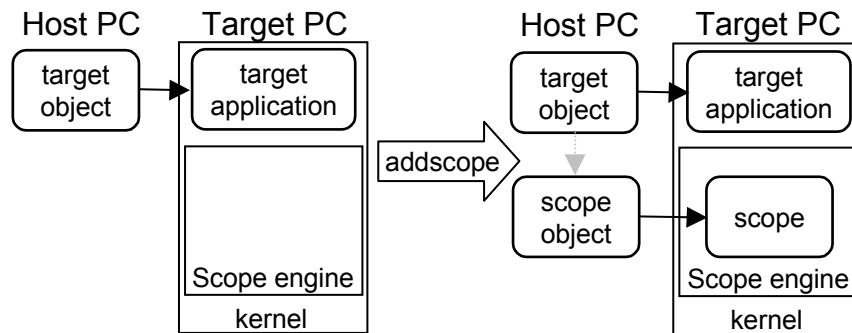
**Description** addscope creates a scope on the target PC, a scope object on the host PC, and updates the target object property Scopes. This method returns a scope object vector. If the result is not assigned to a variable, the scope object properties are listed in the MATLAB window. If you try to add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

A scope acquires data from the target application and displays that data on the target PC, uploads the data to the host PC, or stores that data in a file in the target PC file system.

All scopes of type **target**, **host**, or **file** run on the target PC.

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

**Scope of type host** — Collects data and waits for a command from the host PC for uploading the data. The data is then displayed using a scope viewer on the host or other MATLAB functions.



**Scope of type file** — Data collected is stored in a file in the target PC file system. You can then transfer the data to another PC for examination or plotting.

## Examples

Create a scope and scope object `sc1` using the method `addscope`. A target scope is created on the target PC with an index of 1, and a scope object is created on the host PC, 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 PC with an index of 1, and a scope object is created on the host PC, but it is not assigned to a variable. The target object property `Scopes` is changed from `No scopes defined` to 1.

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



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

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

Create a scope and scope object `sc4` of type `file` using the method `addscope`. A file scope is created on the target PC with an index of 4. A scope object is created on the host PC 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

xPC Target target object methods `remscope` and `getscope`.

xPC Target GUI function `xpcscope`.

xPC Target M-file demo scripts listed in “xPC Target Demos” on page 6-6.

# addsignal

---

**Purpose** Add signals to a scope represented by a scope object

**Syntax** **MATLAB command line**

```
addsignal(scope_object_vector, signal_index_vector)
```

```
scope_object_vector.addsignal(signal_index_vector)
```

**Target command line**

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

**Arguments**

scope\_object\_vector Name of a single scope object or the name of a vector of scope objects.

signal\_index\_vector For one signal, use a single number. For two or more signals, enclose numbers in brackets and separate with commas.

scope\_index Single scope index.

**Description**

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])
```

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

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

# cd

---

**Purpose** Change directory on the target PC

**Syntax** **MATLAB command line**

```
cd(file_obj,target_PC_dir)
file_obj.cd(target_PC_dir)
```

**Arguments**

file_obj	Name of the <code>xpctarget.ftp</code> or <code>xpctarget.fs</code> object.
target_PC_dir	Name of the target PC directory to change to.

**Description** Method of `xpctarget.fsbase`, `xpctarget.ftp`, and `xpctarget.fs` objects. From the host PC, changes directory on the target PC.

**Examples** Change directory from the current to one named `logs` for the file system object `fsys`.

```
cd(fsys,logs) or fsys.cd(logs)
```

Change directory from the current to one named `logs` for the FTP object `f`.

```
cd(f,logs) or f.cd(logs)
```

**See Also** xPC Target file object methods `mkdir` and `pwd`.  
MATLAB `cd` function.

<b>Purpose</b>	Close the serial port connecting the host PC with the target PC
<b>Syntax</b>	<b>MATLAB command line</b> <code>close(target_object)</code> <code>target_object.close</code>
<b>Arguments</b>	<code>target_object</code> Name of a target object.
<b>Description</b>	<code>close</code> closes the serial connection between the host PC and a target PC. If you want to use the serial port for another function without quitting MATLAB — for example, a modem — use this function to close the connection.

# delete

---

**Purpose** Remove target object

**Syntax** **MATLAB command line**  
`delete(target_object)`  
`target_object.delete`

**Arguments** `target_object` Name of a target object.

**Description** Use this method to completely remove the target object. If there are any scopes still associated with the target, this method removes all those scope objects as well.

To ensure that you have successfully removed a target object, type

```
target_object
```

If a message like the following is displayed, you have successfully removed the target object.

```
target_object =  
handle
```

**See Also** The xPC Target target object methods `targetping` and `xpctarget.xpc`.

<b>Purpose</b>	List the contents of the current directory on the target PC
<b>Syntax</b>	<b>MATLAB command line</b> <code>dir(file_obj)</code>
<b>Arguments</b>	<code>file_obj</code> Name of the <code>xpctarget.ftp</code> or <code>xpctarget.fs</code> object.
<b>Description</b>	<p>Method of <code>xpctarget.fsbases</code>, <code>xpctarget.ftp</code>, and <code>xpctarget.fs</code> objects. From the host PC, lists the contents of the current directory on the target PC.</p> <p>To get the results in an M-by-1 structure, use a syntax like <code>ans=dir(file_obj)</code>. This syntax returns a structure like the following:</p> <pre>ans = 1x5 struct array with fields:     name     date     time     bytes     isdir</pre> <p>where</p> <ul style="list-style-type: none"> <li>• <code>name</code> — Name of an object in the directory, 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.</li> <li>• <code>date</code> — Date of the last save of that object</li> <li>• <code>time</code> — Time of the last save of that object</li> <li>• <code>bytes</code> — Size in bytes of that object</li> <li>• <code>isdir</code> — Logical value indicating that the object is (1) or is not (0) a directory</li> </ul>
<b>Examples</b>	<p>List the contents of the current directory for the file system object <code>fsys</code>. You can also list the contents of the current directory for the FTP object <code>f</code>.</p> <pre>dir(fsys) or dir(f)  4/12/1998      20:00                      222390              IO   SYS</pre>

# dir

---

```
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 directory.

## See Also

xPC Target file object methods `mkdir`, `cd`, and `pwd`.

MATLAB `dir` function.



<b>Purpose</b>	Get information about a target PC drive				
<b>Syntax</b>	<b>MATLAB command line</b> <pre>diskinfo(filesys_obj,target_PC_drive) filesys_obj.diskinfo(target_PC_drive)</pre>				
<b>Arguments</b>	<table><tr><td>filesys_obj</td><td>Name of the xpctarget.fs file system object.</td></tr><tr><td>target_PC_drive</td><td>Name of the target PC drive for which to return information.</td></tr></table>	filesys_obj	Name of the xpctarget.fs file system object.	target_PC_drive	Name of the target PC drive for which to return information.
filesys_obj	Name of the xpctarget.fs file system object.				
target_PC_drive	Name of the target PC drive for which to return information.				
<b>Description</b>	Method of xpctarget.fs objects. From the host PC, returns disk information for the specified target PC drive.				
<b>Examples</b>	Return disk information for the target PC C:\ drive for the file system object fsys. <pre>diskinfo(fsys,'C:\') or fsys.diskinfo('C:\')</pre> <pre>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</pre>				

# fclose

---

**Purpose** Close one or more open target PC files (similar to MATLAB `fclose` function)

**Syntax** **MATLAB command line**

```
fclose(filesys_obj,file_ID)
filesys_obj fclose(file_ID)
```

**Arguments**

<code>filesys_obj</code>	Name of the <code>xpctarget.fs</code> file system object.
<code>file_ID</code>	File identifier of the file to close.

**Description** Method of `xpctarget.fs` objects. From the host PC, closes one or more open files in the target PC file system (except standard input, output, and error). The `file_ID` argument is the file identifier associated with an open file (see `fopen` and `filetable`). You cannot have more than eight files open in the file system.

**Examples** Close the open file identified by the file identifier `h` in the file system object `fsys`.

```
fclose(fsys,h) or fsys fclose(h)
```

**See Also** xPC Target file object methods `fopen`, `fread`, `filetable`, and `fwrite`.  
MATLAB `fclose` function.

---

<b>Purpose</b>	Get target PC file information				
<b>Syntax</b>	<b>MATLAB command line</b> <pre>fileinfo(filesys_obj,file_ID) filesys_obj.fileinfo(file_ID)</pre>				
<b>Arguments</b>	<table><tr><td><code>filesys_obj</code></td><td>Name of the <code>xpctarget.fs</code> file system object.</td></tr><tr><td><code>file_ID</code></td><td>File identifier of the file for which to get file information.</td></tr></table>	<code>filesys_obj</code>	Name of the <code>xpctarget.fs</code> file system object.	<code>file_ID</code>	File identifier of the file for which to get file information.
<code>filesys_obj</code>	Name of the <code>xpctarget.fs</code> file system object.				
<code>file_ID</code>	File identifier of the file for which to get file information.				
<b>Description</b>	Method of <code>xpctarget.fs</code> objects. From the host PC, gets the information for the file associated with <code>file_ID</code> .				
<b>Examples</b>	Return file information for the file associated with the file identifier <code>h</code> in the file system object <code>fsys</code> . <pre>fileinfo(fsys,h) or fsys.fileinfo(h)  ans =           FilePos: 0     AllocatedSize: 12288     ClusterChains: 1 VolumeSerialNumber: 1.0450e+009           FullName: 'C:\DATA.DAT'</pre>				

# filetable

---

**Purpose** Get information about open files in the target PC 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 PC, displays a table of the open files in the target PC 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 PC 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. For example,

```
h1 = hex2dec('001E0001')
h1 =
  1966081
```

To close that file, use the `xpctarget.fs fclose` method. For example,

```
fsys fclose(h1);
```

**See Also** `xPC Target file object methods fopen and fclose.`

<b>Purpose</b>	Open a target PC file for reading (similar to MATLAB fopen function)						
<b>Syntax</b>	<b>MATLAB command line</b> <pre>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)</pre>						
<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_name'</code></td><td>Name of the target PC to open.</td></tr><tr><td><code>permission</code></td><td>Values are <code>'r'</code> or <code>'w'</code>. This argument is optional with <code>'r'</code> as the default value.</td></tr></table>	<code>file_obj</code>	Name of the <code>xpctarget.fs</code> object.	<code>'file_name'</code>	Name of the target PC to open.	<code>permission</code>	Values are <code>'r'</code> or <code>'w'</code> . This argument is optional with <code>'r'</code> as the default value.
<code>file_obj</code>	Name of the <code>xpctarget.fs</code> object.						
<code>'file_name'</code>	Name of the target PC to open.						
<code>permission</code>	Values are <code>'r'</code> or <code>'w'</code> . This argument is optional with <code>'r'</code> as the default value.						
<b>Description</b>	<p>Method of <code>xpctarget.fs</code> objects. From the host PC, opens the specified filename on the target PC for binary access.</p> <p>The permission argument values are</p> <ul style="list-style-type: none"><li>• <code>'r'</code> to open the file for reading (default)</li><li>• <code>'w'</code> to open the file for writing. The method creates the file if it does not already exist.</li></ul> <p>You cannot have more than eight files open in the file system. This method returns the file identifier for the open file in <code>file_ID</code>. You use <code>file_ID</code> as the first argument to the other file I/O methods (such as <code>fclose</code>, <code>fread</code>, and <code>fwrite</code>).</p>						
<b>Examples</b>	<p>Open the file <code>data.dat</code> in the target PC file system object <code>fsys</code>. Assign the resulting file handle to a variable for reading.</p> <pre>h = fopen(fsys, 'data.dat') or fsys.fopen('data.dat')  ans =     2883584 d = fread(h);</pre>						
<b>See Also</b>	<code>xPC Target file object methods fclose, fread, and fwrite.</code> MATLAB <code>fopen</code> function.						

# fread

---

**Purpose** Read an open target PC file (similar to MATLAB fread function)

**Syntax** **MATLAB command line**  
fread(file\_obj, file\_ID)  
file\_obj.fread(file\_ID)

**Arguments**

file_obj	Name of the xpctarget.fs object.
file_ID	File identifier of the file to read.

**Description** Method of xpctarget.fs objects. From the host PC, reads the binary data from the file on the target PC and writes it into matrix A. The file\_ID argument is the file identifier associated with an open file (see fopen).

**Examples** Open the file data.dat in the target PC 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 the contents of the file to d. This content is in the xPC Target file format.

**See Also** xPC Target file object methods fclose, fopen, and fwrite.  
MATLAB fread function.

<b>Purpose</b>	Write binary data to an open target PC file (similar to MATLAB fwrite function)						
<b>Syntax</b>	<b>MATLAB command line</b> <pre>fwrite(file_obj,file_ID,A) file_obj.fwrite(file_ID,A)</pre>						
<b>Arguments</b>	<table><tr><td>file_obj</td><td>Name of the xpctarget.fs object.</td></tr><tr><td>file_ID</td><td>File identifier of the file to write.</td></tr><tr><td>A</td><td>Elements of matrix A to be written to the specified file.</td></tr></table>	file_obj	Name of the xpctarget.fs object.	file_ID	File identifier of the file to write.	A	Elements of matrix A to be written to the specified file.
file_obj	Name of the xpctarget.fs object.						
file_ID	File identifier of the file to write.						
A	Elements of matrix A to be written to the specified file.						
<b>Description</b>	Method of xpctarget.fs objects. From the host PC, writes the elements of matrix A to the file identified by file_ID. The data is written to the file in column order. The file_ID argument is the file identifier associated with an open file (see fopen). fwrite requires that the file be open with write permission.						
<b>Examples</b>	<p>Open the file data.dat in the target PC file system object fsys. Assign the resulting file handle to a variable for writing.</p> <pre>h = fopen(fsys,'data.dat','w') or fsys.fopen('data.dat','w') ans =     2883584 d = fwrite(fsys,h,magic(5));</pre> <p>This writes the elements of matrix A to the file handle h. This content is written in column order.</p>						
<b>See Also</b>	xPC Target file object methods fclose, fopen, and fread. MATLAB fwrite function.						

# get (ftp)

---

<b>Purpose</b>	Retrieve a copy of the requested file from the target PC (xpctarget.ftp object)				
<b>Syntax</b>	<b>MATLAB command line</b> <code>get(file_obj, file_name)</code> <code>file_obj.get(file_name)</code>				
<b>Arguments</b>	<table><tr><td><code>file_obj</code></td><td>Name of the xpctarget.ftp object.</td></tr><tr><td><code>file_name</code></td><td>Name of a file on the target PC.</td></tr></table>	<code>file_obj</code>	Name of the xpctarget.ftp object.	<code>file_name</code>	Name of a file on the target PC.
<code>file_obj</code>	Name of the xpctarget.ftp object.				
<code>file_name</code>	Name of a file on the target PC.				
<b>Description</b>	Method of xpctarget.ftp objects. Copies the specified filename from the target PC to the current directory of the host PC. <code>file_name</code> must be either a fully qualified filename on the target PC, or located in the current directory of the target PC.				
<b>Examples</b>	Retrieve a copy of the file named <code>data.dat</code> from the current directory of the target PC file object <code>f</code> .  <code>get(f, 'data.dat')</code> or <code>f.get('data.dat')</code>  <code>ans = data.dat</code>				
<b>See Also</b>	xPC Target file object methods <code>put</code> .				



**Purpose** Return the property values for scope objects

**Syntax** **MATLAB command line**

```
get(scope_object_vector)
```

```
get(scope_object_vector, 'scope_object_property')
```

```
get(scope_object_vector, scope_object_property_vector)
```

**Arguments**

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

**Description** 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.

## get (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.	
AutoRestart	<p>For scopes of type 'File', 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 scope of type 'File' 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, xPC Target overwrites the old data with the new signal data.</p> <p>For scopes of type 'Host' or 'Target', this parameter has no effect.</p>	Yes
Data	<p>Contains the output data for a single data package from a scope.</p> <p>For scopes of type 'Target' or 'File', this parameter has no effect.</p>	
Decimation	A number $n$ , where every $n$ th sample is acquired in a scope window.	Yes

Property	Description	Writable
Filename	<p>Provide a name for the file to contain the signal data. By default, the target PC writes the signal data to a file named <code>C:\data.dat</code> for scope blocks. Note that for scopes of type 'File' created through the MATLAB interface, there is no name initially assigned to <code>FileName</code>. 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, <code>ScopeId</code>, and the beginning letters of the first signal added to the scope.</p> <p>For scopes of type 'Host' or 'Target', this parameter has no effect.</p>	Yes
Grid	<p>Values are 'on' and 'off'.</p> <p>For scopes of type 'Host' or 'File', this parameter has no effect.</p>	Yes

## get (scope object)

Property	Description	Writable
Mode	<p>For scopes of type 'Target', indicate how a scope displays the signals. Values are 'Numerical', 'Redraw' (default), 'Sliding', and 'Rolling'.</p> <p>For scopes of type File, 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 scopes of type Host, this parameter has no effect.</p>	Yes
NumPrePostSamples	<p>For scopes of type 'Host' or 'Target', 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>	

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 scopes of type 'File', 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.	
Signals	List of signal indices from the target object to display on the scope.	
StartTime	<p>Time within the total execution time when a scope begins acquiring a data package.</p> <p>For scopes of type 'Target', this parameter has no effect.</p>	

## get (scope object)

<b>Property</b>	<b>Description</b>	<b>Writable</b>
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'.	
Time	Contains the time data for a single data package from a scope.	
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

Property	Description	Writable
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>	
TriggerScope	<p>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.</p>	Yes
TriggerSignal	<p>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.</p>	Yes

## get (scope object)

Property	Description	Writable
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'.	
WriteSize	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.  If you experience a system crash, you can expect to lose an amount of data the size of WriteSize.  For scopes of type 'Host' or 'Target', this parameter has no effect.	Yes
YLimit	Minimum and maximum y-axis values. This property can be set to 'auto'.  For scopes of type 'Host' or 'File', this parameter has no effect.	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

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

# get (target object)

---

**Purpose** Return the property values for target objects

**Syntax** **MATLAB command line**  
`get(target_object, 'target_object_property')`

**Arguments**  
`target_object` Name of a target object.  
`'target_object_property'` Name of a target object property.

**Description**  
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.	
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.	
Connected	Communication status between the host PC and the target PC. Values are 'Yes' and '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. Correcting CPUoverload requires either a faster processor or a larger sample time.	

Property	Description	Writable
ExecTime	Execution time. Time, in seconds, since your target application started running. When the target application stops, the total execution time is displayed.	
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 menu Configuration Parameters -&gt; xPC Target options</b> pane.	
MaxTET	Maximum task execution time. Corresponds to the slowest time (longest time measured), in seconds, to update model equations and post outputs.	
MinTET	Minimum task execution time. Corresponds to the fastest time (smallest time measured), in seconds, to update model equations and post outputs.	

## get (target object)

<b>Property</b>	<b>Description</b>	<b>Writable</b>
Mode	Type of Real-Time Workshop 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.	
NumLogWraps	The number of times the circular buffer wrapped. The buffer wraps each time the number of samples exceeds MaxLogSamples.	
NumParameters	The number of parameters from your Simulink model that you can tune or change.	
NumSignals	The number of signals from your Simulink model that are available to be viewed with a scope.	
OutputLog	Storage in the MATLAB workspace for the output or y-vector logged during execution of the target application.	

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>	
SampleTime	<p>Time between samples. This value equals the step size, in seconds, for updating the model equations and posting the outputs.</p>	Yes
Scopes	<p>List of index numbers, with one index for each scope.</p>	
SessionTime	<p>Time since the kernel started running on your target PC. This is also the elapsed time since you booted the target PC. Values are in seconds.</p>	
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
ShowSignals	<p>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'.</p>	Yes

## get (target object)

Property	Description	Writable
Signals	<p>List of viewable signals. This list is visible only when ShowSignals is set to 'on'.</p> <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>	
StateLog	<p>Storage in the MATLAB workspace for the state or x-vector logged during execution of the target application.</p>	
Status	<p>Execution status of your target application. Values are 'stopped' and 'running'.</p>	
StopTime	<p>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.</p> <p>When the ExecTime reaches the StopTime, the application stops running.</p>	Yes
TETLog	<p>Storage in the MATLAB workspace for a vector containing task execution times during execution of the target application.</p> <p>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.</p>	

Property	Description	Writable
TimeLog	Storage in the MATLAB workspace for the time or t-vector logged during execution of the target application.	
ViewMode	Display either all scopes or a single scope on the target PC. 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

The xPC Target target object method `set (target object)`.

The scope object methods `get (scope object)` and `set (target object)`.

The built-in MATLAB functions `get` and `set`.

# getfilesize

---

**Purpose** Get the size (in bytes) of a file on the target PC

**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 PC, gets the size (in bytes) of the file identified by the `file_ID` file identifier on the target PC file system.

**Examples** Get the size of the file identifier `h` for the file system object `fsys`.  
`getfilesize(fsys,h)` or `fsys.getfilesize(h)`



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

**Syntax** **MATLAB command line**

```
log = getlog(target_object, 'log_name', first_point,  
number_samples, decimation)
```

<b>Arguments</b>	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, ,2)  
Time_log = getlog(tg, 'TimeLog', 1, ,2)  
plot(Time_log, Output_log)
```

**See Also** xPC Target target object method `get` (target object).

The procedure “Entering the Real-Time Workshop Parameters” on page 3-40.

# getparam

---

**Purpose** Return the value of the specified parameter index from the target object

**Syntax** **MATLAB command line**  
`getparam(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** `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** Get a parameter index or property name from the 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.

**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** The xPC Target scope object method `getsignalid`.

The xPC target M-file demo scripts listed in “xPC Target Demos” on page 6-6.

# getparamname

---

**Purpose** Get the block path and parameter name from the index list

**Syntax** **MATLAB command line**

```
getparamname(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** 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
```

**Purpose** Get a scope object pointing to a scope already defined in the kernel

**Syntax** **MATLAB command line**

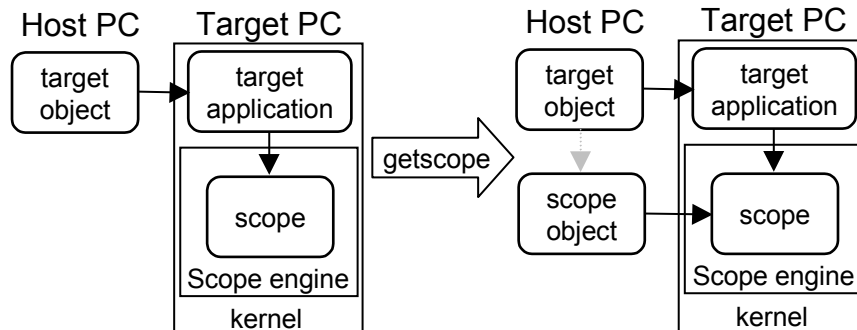
```
scope_object_vector = getscope(target_object, scope_number)

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 scope of type target is created at the time the target application is downloaded to the target PC. To change the number of samples, you need to create a scope object and then change the scope object property NumSamples.

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

## getscope

---

The following example gets the properties of all scopes on the target PC and creates a vector of scope objects on the host PC. If the target object has more than one scope, it create a vector of scope objects.

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

### **See Also**

xPC Target target object methods `getxpcenv` and `remscope`.

xPC target M-file demo scripts listed in “xPC Target Demos” on page 6-6.

**Purpose** Return the value of the specified signal index from the target object

**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.

**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
```

# getsignalid

---

**Purpose** Get the signal index or signal property from the signal list

**Syntax** **MATLAB command line**

```
getsignalid(target_object, 'signal_name')  
tg.getsignalid('signal_name')
```

**Arguments**

target_object	Name of an existing target object.
signal_name	Enter the name of a signal from your Simulink model. For blocks with a single signal, the signal_name is equal to the block_name. For blocks with multiple signals, xPC Target appends S1, S2 ... to the block_name.

**Description** getsignalid returns the index or name of a signal from the signal list, based on the path to the signal name. The block names must be entered in full and are case sensitive.

**Examples** Get the signal index for the single signal from the Simulink block Gain1.

```
getsignalid(tg, 'Gain1') or tg.getsignalid('Gain1')  
ans = 6
```

**See Also** xPC Target target object method getparamid.

xPC Target M-file demo scripts listed in “xPC Target Demos” on page 6-6.



**Purpose** Get the signal name from the index list

**Syntax** **MATLAB command line**

```
getsignalname(target_object, 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
```

# getxpcenv

**Purpose** List environment properties assigned to a MATLAB variable (environment properties)

**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 PC and target PC, the type of C compiler and its location, and the type of target boot floppy created during the setup process. You can view these properties using the getxpcenv function or the xPC Target Explorer. An understanding of the environment properties will help you to correctly configure the xPC Target environment.

<b>Environment Property</b>	<b>Description</b>
Version	xPC Target version number. Read only.
CCompiler	Values are 'Watcom' and 'VisualC'. From the xPC Target Explorer window compiler list, select either Watcom or VisualC.
CompilerPath	Value is a valid compiler root directory. Enter the path where you installed a Watcom C/C++ or Microsoft Visual C/C++ compiler.  If the path is invalid or the directory does not contain the compiler, an error message appears when you use the function updatexpcenv or build a target application.

Environment Property	Description
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 PC. 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 PC. 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 PC 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 PC has more than 64 MB of RAM, and you want to use more than 64 MB, select Manual, and enter the amount of RAM installed in the target PC.</p>
MaxModelSize	<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.</p> <p>Choosing the maximum model size reserves the specified amount of memory on the target PC for the target application. The remaining memory is used by the kernel and by the heap for data logging.</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>
SystemFontSize	<p>Values are 'Small' and 'Large'.</p>

Environment Property	Description
CANLibrary	<p>Values are 'None', '200 ISA', '527 ISA', '1000 PCI', '1000 MB PCI', and 'PC104'.</p> <p>From the xPC Target Explorer window <b>CAN Library</b> list, select None, 200 ISA, 527 ISA, 1000 PCI, 1000 MB PCI, or PC104.</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>
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. xPC Target automatically determines the COM port on the target PC.</p> <p>Before you can select an RS-232 port, you need to set the HostTargetComm property to RS232.</p>
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>

<b>Environment Property</b>	<b>Description</b>
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 PC. Ask your system administrator for this value.</p> <p>For example, 192.168.0.1.</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 PC.</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
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 PC.</p> <p>If you communicate with your target PC from within a LAN that uses gateways, and your host and target computers are connected through a gateway, then you need to enter a value for this property. If your LAN does not use gateways, you do not need to change this property. Ask your system administrator.</p>
TcpIpTargetDriver	<p>Values are 'NE2000', 'SMC91C9X', 'I82559', and 'RTLANCE'.</p> <p>From the xPC Target Explorer window <b>TcpIp target driver</b> list, select NE2000, SMC91C9X, I82559, or RTLANCE. The Ethernet card provided with xPC Target uses the NE2000 driver.</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 PC. You do not need to define a bus type for your host PC, which can be the same or different from the bus type in your target PC.</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
TcpIpTargetISAMemPort	<p>Value is '0xn timer'.</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 PC, choose another I/O port base address and make the corresponding changes to your jumper settings.</p>
TcpIpTargetISAIRQ	<p>Value is 'n', where <math>n</math> 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>The MathWorks recommends setting the IRQ to 5, 10, or 11. If one of these hardware settings leads to a conflict in your target PC, choose another IRQ and make the corresponding changes to your jumper settings.</p>

Environment Property	Description
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 PC displays information as text.</p> <p>To use all the features of the target scope, you also need to install a keyboard and mouse on the target PC.</p>
TargetMouse	<p>Values are 'None', 'PS2', 'RS232 COM1', and 'RS232 COM2'.</p> <p>From the xPC Target Explorer window <b>Target mouse</b> list, select None, PS2, RS232 COM1, or RS232 COM2.</p> <p>Before you can select a target mouse, you need to set the Target Scope property to Enabled.</p> <p><b>Target mouse</b> allows you to disable or enable mouse support on the target PC:</p> <ul style="list-style-type: none"><li>• If you do not connect a mouse to the target PC, you need to set this property to None; otherwise, the target application might not behave properly.</li><li>• If the target PC supports PS/2 devices (keyboard and mouse) and you connect a PS/2 mouse, set this property to PS2.</li><li>• If you connect a serial RS-232 mouse to the target PC, select either RS232 COM1 or RS232 COM2 depending on which serial port you attached the mouse to.</li></ul>



Environment Property	Description
EmbeddedOption	<p>Values are 'Disabled' and 'Enabled'. This property is read only.</p> <p>Note that the xPC Target Embedded Option is enabled only if you purchase an additional license.</p>
TargetBoot	<p>Values are 'BootFloppy', 'DOSLoader', and 'StandAlone'.</p> <p>From the xPC Target Explorer window <b>Target boot mode</b> list, select BootFloppy, DOSLoader, or StandAlone.</p> <p>If your license file does not include the license for the xPC Target Embedded Option, the <b>Target boot mode</b> box is disabled, with BootFloppy as your only selection. With the xPC Target Embedded Option licensed and installed, you have the additional choices of DOSLoader and StandAlone.</p>

## 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

xPC Target functions setxpcenv, updatexpcenv, xpcbootdisk, and xpcsetup

# getxpcpci

---

**Purpose** Determine which PCI boards are installed in the target PC

**Syntax** **MATLAB Command Line**

```
getxpcpci(target_object, 'type_of_boards')
```

**Arguments**

target_object	Variable name to reference the target object.
type_of_boards	Values are no arguments, 'all', and 'supported'.

**Description** The `getxpcpci` function displays, in the MATLAB window, which PCI boards are installed in the target PC. By default, `getxpcpci` displays this information for the target object, `tg`. If you have multiple target PCs 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, and the board PCI ID itself.

For a successful query,

- The host-target communication link must be working. (The function `xpctargetping` must return success before 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 prior to the model build process.

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

```
pcidevs = getxpcpci
```

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

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

Display a list of the currently supported PCI devices in the xPC Target block library. The result is displayed.

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

When called with the 'supported' option, `getxpcpci` does not access the target PC.

To display the list of PCI devices installed on the target PC, `tg1`, first create a target object, `tg1`, for that target PC. Then, call `getxpcpci` with the 'all' option. For example

```
tg1=xpctarget.xpc('RS232','COM1','115200')  
getxpcpci(tg1, 'all')
```

# load

---

**Purpose** Download a target application to the target PC

**Syntax** **MATLAB command line**

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

**Arguments**

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

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

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

**Examples** Load the target application xpcosc represented by the target object tg.

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

**See Also** xPC Target function unload.

xPC Target M-file demo scripts listed in “xPC Target Demos” on page 6-6.

**Purpose** Restore the parameter values saved in the specified file

**Syntax** **MATLAB command line**

```
loadparamset(target_object, 'filename')  
target_object.loadparamset('filename')
```

<b>Arguments</b>	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 PC. This method assumes that you have a parameter file from a previous run of the saveparamset method.

**See Also** xPC Target target object method saveparamset.

# mkdir

---

<b>Purpose</b>	Make a directory on the target PC				
<b>Syntax</b>	<b>MATLAB command line</b> <pre>mkdir(file_obj,dir_name) file_obj.mkdir(dir_name)</pre>				
<b>Arguments</b>	<table><tr><td>file_obj</td><td>Name of the <code>xpctarget.ftp</code> or <code>xpctarget.fs</code> object.</td></tr><tr><td>dir_name</td><td>Name of the directory to be created.</td></tr></table>	file_obj	Name of the <code>xpctarget.ftp</code> or <code>xpctarget.fs</code> object.	dir_name	Name of the directory to be created.
file_obj	Name of the <code>xpctarget.ftp</code> or <code>xpctarget.fs</code> object.				
dir_name	Name of the directory to be created.				
<b>Description</b>	<p>Method of <code>xpctarget.fsbases</code>, <code>xpctarget.ftp</code>, and <code>xpctarget.fs</code> objects. From the host PC, makes a new directory in the current directory on the target PC file system.</p> <p>Note that to delete a directory from the target PC, you need to reboot the PC into DOS or some other operating system and use a utility in that system to delete the directory.</p>				
<b>Examples</b>	<p>Create a new directory, <code>logs</code>, in the target PC file system object <code>fsys</code>.</p> <pre>mkdir(fsys,logs) or fsys.mkdir(logs)</pre> <p>Create a new directory, <code>logs</code>, in the target PC FTP object <code>f</code>.</p> <pre>mkdir(f,logs) or f.mkdir(logs)</pre>				
<b>See Also</b>	<p>xPC Target file object methods <code>dir</code> and <code>pwd</code>.</p> <p>MATLAB <code>mkdir</code> function.</p>				

---

<b>Purpose</b>	Copy a file from the host PC to the target PC				
<b>Syntax</b>	<b>MATLAB command line</b> <pre>put(file_obj,file_name) file_obj.put(file_name)</pre>				
<b>Arguments</b>	<table><tr><td><code>file_obj</code></td><td>Name of the <code>xpctarget.ftp</code> object.</td></tr><tr><td><code>file_name</code></td><td>Name of the file to copy to the target PC.</td></tr></table>	<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 PC.
<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 PC.				
<b>Description</b>	<p>Method of <code>xpctarget.ftp</code> objects. Copies a file from the host PC to the target PC. <code>file_name</code> must be a file in the current directory of the host PC. The method writes <code>file_name</code> to the target PC disk.</p> <p><code>put</code> might be slower than the <code>get</code> operation for the same file. This is expected behavior.</p>				
<b>Examples</b>	<p>Copy the file <code>data2.dat</code> from the current directory of the host PC to the current directory of the target PC FTP object <code>f</code>.</p> <pre>put(f,'data2.dat') or fsys.put('data2.dat')</pre>				
<b>See Also</b>	<code>xPC Target file object methods <code>dir</code> and <code>get (ftp)</code>.</code>				

# pwd

---

<b>Purpose</b>	Retrieve the current directory path of the target PC		
<b>Syntax</b>	<b>MATLAB command line</b> <pre>pwd(file_obj) file_obj.pwd</pre>		
<b>Arguments</b>	<table><tr><td>file_obj</td><td>Name of the xpctarget.ftp or xpctarget.fs object.</td></tr></table>	file_obj	Name of the xpctarget.ftp or xpctarget.fs object.
file_obj	Name of the xpctarget.ftp or xpctarget.fs object.		
<b>Description</b>	Method of xpctarget.fsbase, xpctarget.ftp, and xpctarget.fs objects. Returns the pathname of the current target PC directory.		
<b>Examples</b>	Return the target PC current directory for the file system object fsys. <pre>pwd(fsys) or fsys.pwd</pre> Return the target PC current directory for the FTP object f. <pre>pwd(f) or f.pwd</pre>		
<b>See Also</b>	xPC Target file object methods dir and mkdir. MATLAB pwd function.		



**Purpose** Reboot the target PC

**Syntax** **MATLAB command line**  
`reboot(target_object)`

**Target PC command line**  
`reboot`

**Arguments** `target_object` Name of an existing target object.

**Description** `reboot` reboots the target PC, 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 PC back to Windows after removing the target boot disk.

---

**Note** This method might not work on some target hardware.

---

**See Also** xPC Target target object methods `load` and `unload`.

# remscope

**Purpose** Remove a scope from the target PC

**Syntax** **MATLAB command line**

```
remscope(target_object, scope_number_vector)  
target_object.remscope(scope_number_vector)
```

```
remscope(target_object)  
target_object.remscope
```

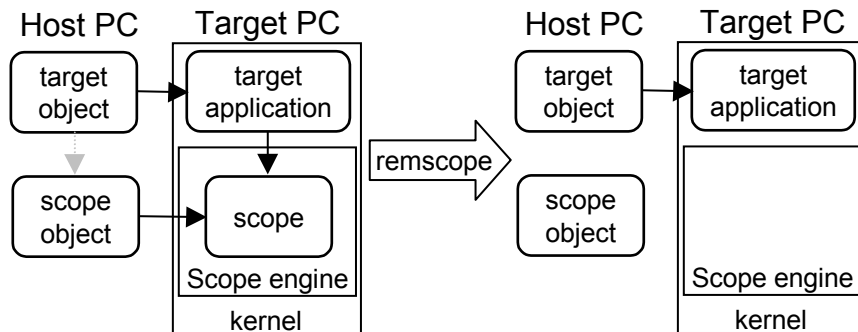
**Target PC command line**

```
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 PC. The method remscope has no return value. The scope object representing the scope on the host PC 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**

xPC Target target object methods `addscope` and `getscope`.

xPC target M-file demo scripts listed in “xPC Target Demos” on page 6-6.

# removefile

---

**Purpose** Remove a file from the target PC

**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 PC file system.
file_obj	Name of the xpctarget.fs object.

**Description** Method of xpctarget.fs objects Removes a file from the target PC file system.

---

**Note** You cannot recover this file once it is removed.

---

**Examples** Remove the file data2.dat from the target PC file system fsys.

```
removefile(fsys,'data2.dat') or fsys.removefile('data2.dat')
```

**Purpose** Remove signals from a scope represented by a scope object

**Syntax** **MATLAB command line**

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

**Target command line**

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

**Arguments**

scope_object	MATLAB object created with the target object 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
```

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

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

# remsignal

---

## **See Also**

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

**Purpose** Remove a directory from the target PC

**Syntax** **MATLAB command line**

```
rmdir(file_obj,dir_name)  
file_obj.rmdir(dir_name)
```

**Arguments**

<code>dir_name</code>	Name of the directory to remove from the target PC file system.
<code>file_obj</code>	Name of the <code>xpctarget.fs</code> object.

**Description** Method of `xpctarget.fsbases`, `xpctarget.ftp`, and `xpctarget.fs` objects. Removes a directory from the target PC file system.

---

**Note** You cannot recover this directory once it is removed.

---

**Examples** Remove the directory `data2dir.dat` from the target PC file system `fsys`.

```
rmdir(f,'data2dir.dat') or fsys.rmdir('data2dir.dat')
```

# saveparamset

---

**Purpose** Save the parameter values of the current target application

**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 PC (C:\ by default). You can later reload these parameters with the 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** xPC Target target object method loadparamset.



**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, . . .)
```

<b>Arguments</b>	scope_object	Name of a scope object or a vector of scope objects.
	'property_name'	Name of a scope object property. Always use quotation marks.
	property_value	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.

## 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.	
AutoRestart	<p>For scopes of type 'File', 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 scope of type 'File' 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, xPC Target overwrites the old data with the new signal data.</p> <p>For scopes of type 'Host' or 'Target', this parameter has no effect.</p>	Yes
Data	<p>Contains the output data for a single data package from a scope.</p> <p>For scopes of type 'Target' or 'File', this parameter has no effect.</p>	
Decimation	A number $n$ , where every $n$ th sample is acquired in a scope window.	Yes

Property	Description	Writable
Filename	<p>Provide a name for the file to contain the signal data. By default, the target PC writes the signal data to a file named <code>C:\data.dat</code> for scope blocks. Note that for scopes of type 'File' created through the MATLAB interface, there is no name initially assigned to <code>FileName</code>. 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, <code>ScopeId</code>, and the beginning letters of the first signal added to the scope.</p> <p>For scopes of type 'Host' or 'Target', this parameter has no effect.</p>	Yes
Grid	<p>Values are 'on' and 'off'.</p> <p>For scopes of type 'Host' or 'File', this parameter has no effect.</p>	Yes

## set (scope object)

Property	Description	Writable
Mode	<p>For scopes of type 'Target', indicate how a scope displays the signals. Values are 'Numerical', 'Redraw' (default), 'Sliding', and 'Rolling'.</p> <p>For scopes of type File, 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 scopes of type Host, this parameter has no effect.</p>	Yes
NumPrePostSamples	<p>For scopes of type 'Host' or 'Target', 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>	

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 scopes of type 'File', 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.	
Signals	List of signal indices from the target object to display on the scope.	
StartTime	<p>Time within the total execution time when a scope begins acquiring a data package.</p> <p>For scopes of type 'Target', this parameter has no effect.</p>	

## set (scope object)

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'.	
Time	Contains the time data for a single data package from a scope.	
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

Property	Description	Writable
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>	
TriggerScope	<p>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.</p>	Yes
TriggerSignal	<p>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.</p>	Yes

## set (scope object)

Property	Description	Writable
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'.	
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 scopes of type 'Host' or 'Target', 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 scopes of type 'Host' or 'File', this parameter has no effect.</p>	Yes

### 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

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

# set (target object)

---

**Purpose** Change property values for target objects

**Syntax** **MATLAB command line**

```
set(target_object)
```

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

```
target_object.set('property_name1', 'property_value1')
```

```
set(target_object, property_name_vector, property_value_vector)
```

```
target_object.property_name = property_value
```

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

```
parameter_name = parameter_value
```

```
stoptime = floating_point_number
```

```
sampletime = floating_point_number
```

**Arguments**

<code>target_object</code>	Name of a target object.
<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.
<code>parameter_name</code>	The letter p followed by the parameter index. For example, p0, p1, p2.

**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 are listed in the following table. This table includes a description of the properties:

<b>Property</b>	<b>Description</b>	<b>Writable</b>
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.	Yes
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

## set (target object)

<b>Property</b>	<b>Description</b>	<b>Writable</b>
StopTime	<p>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.</p> <p>When the ExecTime reaches the StopTime, the application stops running.</p>	Yes
ViewMode	<p>Display either all scopes or a single scope on the target PC. Value is 'all' or a single scope index. This property is active only if the environment property TargetScope is set to enabled.</p>	Yes

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

xPC Target target object method `get` (target object).

Scope object methods `get` (scope object) and `set` (scope object).

Built in MATLAB functions `get` and `set`.

xPC target M-file demo scripts listed in “xPC Target Demos” on page 6-6.

# setparam

---

**Purpose** Set writable target object parameters to the specified values

**Syntax** **MATLAB command line**

```
setparam(target_object, 'parameter_value')
```

**Arguments**

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

**Purpose** Change xPC Target environment properties (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. Use the function `updatexpcenv` to change the current properties to the new properties.

The environment properties define communication between the host PC and target PC, the type of C compiler and its location, and the type of target boot floppy created during the setup process. With the exception of the `Version` property, you can set these properties using the `xpcexplr` function or the xPC Target Explorer window. An understanding of the environment properties will help you to correctly configure the xPC Target environment.

Environment Property	Description
Version	xPC Target version number. Read only.
CCompiler	Values are 'Watcom' and 'VisualC'. From the xPC Target Explorer window compiler list, select either Watcom or VisualC.

Environment Property	Description
CompilerPath	<p>Value is a valid compiler root directory. Enter the path where you installed a Watcom C/C++ or Microsoft Visual C/C++ compiler.</p> <p>If the path is invalid or the directory does not contain the compiler, an error message appears when you use the function <code>updatexpcenv</code> or build a target application.</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 PC. 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 PC. 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 PC 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 PC has more than 64 MB of RAM, and you want to use more than 64 MB, select Manual, and enter the amount of RAM installed in the target PC.</p>



Environment Property	Description
MaxModelSize	<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.</p> <p>Choosing the maximum model size reserves the specified amount of memory on the target PC for the target application. The remaining memory is used by the kernel and by the heap for data logging.</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>
SystemFontSize	<p>Values are 'Small' and 'Large'.</p>
CANLibrary	<p>Values are 'None', '200 ISA', '527 ISA', '1000 PCI', '1000 MB PCI', and 'PC104'.</p> <p>From the xPC Target Explorer window <b>CAN Library</b> list, select None, 200 ISA, 527 ISA, 1000 PCI, 1000 MB PCI, or PC104.</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>

Environment Property	Description
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. xPC Target automatically determines the COM port on the target PC.</p> <p>Before you can select an RS-232 port, you need to set the HostTargetComm property to RS232.</p>
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>
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 PC. Ask your system administrator for this value.</p> <p>For example, 192.168.0.1.</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 PC.</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>
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 PC.</p> <p>If you communicate with your target PC from within a LAN that uses gateways, and your host and target computers are connected through a gateway, then you need to enter a value for this property. If your LAN does not use gateways, you do not need to change this property. Ask your system administrator.</p>
TcpIpTargetDriver	<p>Values are 'NE2000', 'SMC91C9X', 'I82559', and 'RTLANCE'.</p> <p>From the xPC Target Explorer window <b>TcpIp target driver</b> list, select NE2000, SMC91C9X, I82559, or RTLANCE. The Ethernet card provided with xPC Target uses the NE2000 driver.</p>

Environment Property	Description
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 PC. You do not need to define a bus type for your host PC, which can be the same or different from the bus type in your target PC.</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>
TcpIpTargetISAMemPort	<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 PC, choose another I/O port base address and make the corresponding changes to your jumper settings.</p>

Environment Property	Description
TcpIpTargetISAIRQ	<p>Value is 'n', where <math>n</math> 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>The MathWorks recommends setting the IRQ to 5, 10, or 11. If one of these hardware settings leads to a conflict in your target PC, choose another IRQ and make the corresponding changes to your jumper settings.</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 PC displays information as text.</p> <p>To use all the features of the target scope, you also need to install a keyboard and mouse on the target PC.</p>

Environment Property	Description
TargetMouse	<p>Values are 'None ', 'PS2 ', 'RS232 COM1 ', and 'RS232 COM2 '.</p> <p>From the xPC Target Explorer window <b>Target mouse</b> list, select None, PS2, RS232 COM1, or RS232 COM2.</p> <p>Before you can select a target mouse, you need to set the Target Scope property to Enabled.</p> <p><b>Target mouse</b> allows you to disable or enable mouse support on the target PC:</p> <ul style="list-style-type: none"><li>• If you do not connect a mouse to the target PC, you need to set this property to None; otherwise, the target application might not behave properly.</li><li>• If the target PC supports PS/2 devices (keyboard and mouse) and you connect a PS/2 mouse, set this property to PS2.</li><li>• If you connect a serial RS-232 mouse to the target PC, select either RS232 COM1 or RS232 COM2 depending on which serial port you attached the mouse to.</li></ul>

Environment Property	Description
EmbeddedOption	<p>Values are 'Disabled' and 'Enabled'. This property is read only.</p> <p>Note that the xPC Target Embedded Option is enabled only if you purchase an additional license.</p>
TargetBoot	<p>Values are 'BootFloppy', 'DOSLoader', and 'StandAlone'.</p> <p>From the xPC Target Explorer window <b>Target boot mode</b> list, select BootFloppy, DOSLoader, or StandAlone.</p> <p>If your license file does not include the license for the xPC Target Embedded Option, the <b>Target boot mode</b> box is disabled, with BootFloppy as your only selection. With the xPC Target Embedded Option licensed and installed, you have the additional choices of DOSLoader and StandAlone.</p>

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 PC to COM2.

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

## See Also

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

Graphical Interface” on page 6-3 and “Changing Environment Properties with a Command-Line Interface” on page 6-5.



**Purpose** Start execution of a scope on a target PC

**Syntax** **MATLAB command line**

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

```
start(getscope((target_object, signal_index_vector))
```

**Target PC command line**

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

**Arguments**

target_object	Name of a target object.
scope_object_vector	Name of a single scope object, name of vector of scope objects, list of scope object names in 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 PC represented by a scope object on the host PC. This method does not necessarily start data acquisition, which depends on the trigger settings. Before using this method, 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.

## start (scope object)

---

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

or type

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

or type

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

Start all scopes:

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

or type

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

### See Also

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

<b>Purpose</b>	Start execution of a target application on a target PC
<b>Syntax</b>	<b>MATLAB command line</b> <pre>start(target_object) target_object.start +target_object</pre> <b>Target PC command line</b> <pre>start</pre>
<b>Arguments</b>	target_object                      Name of a target object. The default name is tg.
<b>Description</b>	Method of both target and scope objects. Starts execution of the target application represented by the target object. Before using this method, the target application must be created and loaded on the target PC. If a target application is running, this command has no effect.
<b>Examples</b>	Start the target application represented by the target object tg. <pre>+tg tg.start start(tg)</pre>
<b>See Also</b>	xPC Target target object methods stop (target object), load, and unload. Scope object method stop (scope object).

# stop (scope object)

---

**Purpose** Stop execution of a scope on the target PC

**Syntax** **MATLAB command line**

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

```
stop(getscope(target_object, signal_index_vector))
```

**Target PC command line**

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

**Arguments**

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

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

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

or type

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

**See Also**

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

# stop (target object)

---

<b>Purpose</b>	Stop execution of a target application on a target PC
<b>Syntax</b>	<b>MATLAB command line</b>  stop(target_object) target_object.stop -target_object  <b>Target PC command line</b>  stop
<b>Arguments</b>	target_object                      Name of a target object.
<b>Description</b>	Stops execution of the target application represented by the target object. If the target application is stopped, this command has no effect.
<b>Examples</b>	Stop the target application represented by the target object tg.  stop(tg) or tg.stop or -tg
<b>See Also</b>	The xPC Target target object method start (target object). The scope object methods stop (scope object) and start (scope object).

<b>Purpose</b>	Test communication between a host and its target computers
<b>Syntax</b>	<b>MATLAB command line</b> <code>targetping(target_object)</code> <code>target_object.targetping</code>
<b>Arguments</b>	<code>target_object</code> Name of a target object.
<b>Description</b>	Method of a target object. Use this method to ping a target PC from the host PC. It returns either <code>success</code> or <code>failed</code> . If the xPC Target kernel is loaded, running, and communication is working properly, this function returns the value <code>success</code> .  This function works with both RS-232 and TCP/IP communication.
<b>Examples</b>	Ping the communication between the host and the target object <code>tg</code> . <code>targetping(tg)</code> or <code>tg.targetping</code>
<b>See Also</b>	The xPC Target target object methods <code>delete</code> and <code>xpctarget.xpc</code> .

# trigger

---

**Purpose** Software-trigger the start of data acquisition for one or more scopes

**Syntax** **MATLAB command line**

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

**Arguments**

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

**Description** Method for a scope object. If the scope object property TriggerMode has a value of 'software', 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.

**Examples** 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)
```



<b>Purpose</b>	Remove the current target application from the target PC
<b>Syntax</b>	<b>MATLAB command line</b> <code>unload(target_object)</code> <code>target_object.unload</code>
<b>Arguments</b>	<code>target_object</code> Name of a target object that represents a target application.
<b>Description</b>	Method of a target object. The kernel goes into loader mode and is ready to download new target application from the host PC.
<b>Examples</b>	Unload the target application represented by the target object <code>tg</code> . <code>unload(tg)</code> or <code>tg.unload</code>
<b>See Also</b>	xPC Target methods <code>load</code> and <code>reboot</code> .

# updatexpcenv

---

<b>Purpose</b>	Change current environment properties to equal new properties (environment properties)
<b>Syntax</b>	<b>MATLAB Command Line</b> updatexpcenv
<b>Description</b>	Function to update environment properties. Call the function updatexpcenv in the following order: <ol style="list-style-type: none"><li>1 Enter new properties with the function setxpcenv.</li><li>2 Type updatexpcenv to change the current properties to match the new properties.</li><li>3 Create a target boot floppy with the function xpcbootdisk.</li></ol>
<b>See Also</b>	The xPC Target functions setxpcenv, getxpcenv, xpcbootdisk, and xpcsetup. The procedures “Changing Environment Properties with a Graphical Interface” on page 6-3 and “Changing Environment Properties with a Command-Line Interface” on page 6-5.

**Purpose** Call the target object constructor, `xpctarget.xpc`

**Note** See “xpctarget.xpc” on page 16-117

# xpcbootdisk

---

**Purpose** Create xPC Target boot disk and confirm the current environment properties (environment properties)

**Syntax** **MATLAB Command Line**  
xpcbootdisk

**Description** Function to create an xPC target boot floppy for the current xPC Target environment that has been updated with the function `updatexpcenv`. Creating an xPC Target boot floppy consists of writing the correct bootable kernel image onto the disk. You are asked to insert an empty formatted floppy disk into the 3.5 inch disk drive.

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

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

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

**See Also** The xPC Target functions `setxpcenv`, `getxpcenv`, `updatexpcenv`, and `xpcsetup`. The procedures “Changing Environment Properties with a Graphical Interface” on page 6-3 and “Changing Environment Properties with a Command-Line Interface” on page 6-5.

<b>Purpose</b>	Open the <b>xPC Target Explorer</b> window
<b>Syntax</b>	<b>MATLAB Command Line</b> <code>xpcexplr</code>
<b>Description</b>	This graphical user interface (GUI) allows you to <ul style="list-style-type: none"><li>• Manage an xPC Target system</li><li>• Enter and change environment properties</li><li>• Create an xPC Target boot disk</li><li>• Build, download, and run target applications</li><li>• Monitor signals</li><li>• Tune parameters</li></ul>
<b>See Also</b>	The xPC Target functions <code>setxpcenv</code> , <code>getxpcenv</code> , <code>updatexpcenv</code> , and <code>xpcbootdisk</code> . The procedures “Environment Properties for Serial Communication” on page 2-21 and “Environment Properties for Network Communication” on page 2-33.

# xpcrctool

---

<b>Purpose</b>	Open the remote control tool on the host PC
<b>Syntax</b>	<b>MATLAB Command Line</b> <code>xpcrctool</code>
<b>Description</b>	This graphical user interface (GUI) allows you to control your target application running on the target PC. It also allows you to add scopes, acquire signal data, and tune parameters using the xPC Target Simulink browser.
<b>See Also</b>	The xPC Target functions <code>xpcscope</code> , <code>xpctgscope</code> , and the procedure “Signal Tracing” on page 3-7

<b>Purpose</b>	Open a scope manager window on the host PC
<b>Syntax</b>	<b>MATLAB Command Line</b> <code>xpcscope</code>
<b>Description</b>	This graphical user interface (GUI) allows you to define scopes that are displayed on your host PC, choose signals, and control the data acquisition process.
<b>See Also</b>	The xPC Target function <code>xpctgscope</code> and the procedure “Signal Tracing” on page 3-7

# xpcsetup

---

**Purpose** Open the **Setup** window

**Syntax** **MATLAB Command Line**  
xpcsetup

**Description** This graphical user interface (GUI) allows you to

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

**See Also** The xPC Target functions `setxpcenv`, `getxpcenv`, `updatexpcenv`, and `xpcbootdisk`. The procedures “Environment Properties for Serial Communication” on page 2-21 and “Environment Properties for Network Communication” on page 2-33.



**Purpose** Create a target object representing the target application

**Syntax** **MATLAB command line**

```
target_object = xpctarget.xpc('mode', 'arg1', 'arg2')
```

<b>Arguments</b>	target_object	Variable name to reference the target object:
	mode	Optionally, enter the communication mode
	TCP/IP	Enable TCP/IP connection with target PC.
	RS232	Enable RS-232 connection with target PC.
	arg1	Optionally, enter an argument based on the mode value:
	IP address	If mode is 'TCP/IP', enter the IP address of the target PC.
	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 PC.
	Baud rate	If mode is 'RS232', enter the baud rate for the connection between the host and target PC.

**Description** Constructor of a target object. The target object represents the target application and target PC. You make changes to the target application by changing the target object using methods and properties.

If you have one target PC, or if you designate a target PC as the default one in your system, use the syntax

```
target_object=xpctarget.xpc
```

If you have multiple target PCs in your system, use the following syntax to create the additional target objects.

```
target_object=xpctarget.xpc('mode', 'arg1', 'arg2')
```

## 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 `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
```

## See Also

xPC Target methods `get (target object)`, `set (target object)`, `delete`, and `targetping`.

<b>Purpose</b>	Test communication between the host and target computers
<b>Syntax</b>	<b>MATLAB Command Line</b> <code>xpctargetping</code>
<b>Examples</b>	Check for communication between the host PC and target PC. <code>xpctargetping</code>
<b>Description</b>	<p>Pings the target PC from the host PC and returns either success or failed. If the xPC Target kernel is loaded, running, and communication is working properly, this function returns the value success.</p> <p>This function works with both RS-232 and TCP/IP communication.</p> <pre>ans = success</pre>
<b>See Also</b>	The xPC Target procedure “Testing and Troubleshooting the Installation” on page 2-45

# xpctargetspy

---

<b>Purpose</b>	Open a <b>Real-Time xPC Target Spy</b> window on the host PC
<b>Syntax</b>	<b>MATLAB Command Line</b> <code>xpctargetspy(target_object)</code>
<b>Arguments</b>	<code>target_object</code> Variable name to reference the target object.
<b>Description</b>	<p>This graphical user interface (GUI) allows you to upload displayed data from the target PC. By default, <code>xpctargetspy</code> opens a <b>Real-Time xPC Target Spy</b> window for the target object, <code>tg</code>. If you have multiple target PCs in your system, you can call the <code>xpctargetspy</code> function for a particular target object, <code>target_object</code>.</p> <p>The behavior of this function depends on the value for the environment property <code>TargetScope</code>:</p> <ul style="list-style-type: none"><li>• If <code>TargetScope</code> is enabled, a single graphics screen is uploaded. The screen is not continually updated because of a higher data volume when a target graphics card is in VGA mode. To update the host screen with another target screen, move the pointer into the <b>Real-Time xPC Target Spy</b> window and left-click.</li><li>• If <code>TargetScope</code> is disabled, text output is transferred once every second to the host and displayed in the window.</li></ul>
<b>Examples</b>	<p>To open the <b>Real-Time xPC Target Spy</b> window for a default target PC, <code>tg</code>, in the MATLAB window, type</p> <pre>xpctargetspy</pre> <p>To open the <b>Real-Time xPC Target Spy</b> window for a target PC, <code>tg1</code>, in the MATLAB window, type</p> <pre>xpctargetspy(tg1)</pre>

---

<b>Purpose</b>	Test the xPC Target installation
<b>Syntax</b>	<b>MATLAB Command Line</b> <pre>xpctest xpctest('noreboot')</pre>
<b>Arguments</b>	'noreboot'                      Only one possible option. Skips the reboot test. Use this option if the target hardware does not support software rebooting. Value is 'noreboot'.
<b>Description</b>	<p>xpctest is a series of xPC Target tests to check the correct functioning of the following xPC Target tasks:</p> <ul style="list-style-type: none"><li>• Initiate communication between the host and target computers.</li><li>• Reboot the target PC to reset the target environment.</li><li>• Build a target application on the host PC.</li><li>• Download a target application to the target PC.</li><li>• Check communication between the host and target computers using commands.</li><li>• Execute a target application.</li><li>• Compare the results of a simulation and the target application run.</li></ul> <p>xpctest('noreboot') skips the reboot test. Use this option if target hardware does not support software rebooting.</p>
<b>Examples</b>	<p>If the target hardware does not support software rebooting, or to skip the reboot test, in the MATLAB window, type</p> <pre>xpctest('noreboot')</pre>
<b>See Also</b>	Procedures “Testing and Troubleshooting the Installation” on page 2-45 and “Test 1, Ping Target System Standard Ping” on page 2-47

# xpctgscope

---

<b>Purpose</b>	Open the <b>xPC Target: Target Manager</b> window
<b>Syntax</b>	<b>MATLAB Command Line</b> <code>xpctgscope</code>
<b>Description</b>	This graphical user interface (GUI) allows you to define scopes that are displayed on your target PC, choose signals, and control the data acquisition process.
<b>See Also</b>	The xPC Target function <code>xpcscope</code> and the procedure “Signal Tracing” on page 3-7

**Purpose** Disconnect the target PC from the current client application

**Syntax** **MATLAB Command Line**  
`xpcwwenable`

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





## A

- applications
  - with DOSLoader mode 5-16
  - with StandAlone mode 5-21

## B

- block parameters
  - parameter tuning with external mode 3-40
  - parameter tuning with xpcrcctool 15-26

## C

- C compiler
  - setup dialog box 15-2
- changing environment properties 6-5
- changing parameters
  - using target object properties 3-38
  - xPC Target commands 3-38
- changing properties
  - environment properties with xPC Target Explorer 6-3
- changing properties with xPC Target Setup
  - environment properties 15-8
- command-line interface
  - aliasing 14-10
  - scope object 1-3
  - scope object methods 14-5
  - scope object property commands 14-8
  - target object methods 14-3
  - target object property commands 14-3
  - target objects 1-2
  - target PC 7-1
- controlling target applications
  - with xpcrcctool 15-12
- creating
  - scope objects 15-14

- creating application with DOSLoader mode 5-16
- creating application with StandAlone mode 5-21
- custom GUI 9-8
  - Dials & Gauges Blockset 9-8

## D

- data logging
  - with MATLAB 3-30
  - with remote control tool 15-25
  - with Web browser 3-34
- DOSLoader mode 5-3
  - copying kernel 5-14
  - creating target application 5-16

## E

- embedded option
  - DOSLoader 5-3
  - introduction 5-2
  - StandAlone 5-3
  - updating xPC Target environment 5-9
- entering environment properties
  - xPC Target Explorer 6-3
  - xPC Target Setup 15-8
- environment
  - network communication 15-4
  - serial communication 15-2

- environment properties
  - and StandAlone mode 5-17
  - changing through CLI 6-5
  - changing through xPC Target Explorer 6-3
  - changing through xPC Target Setup 15-8
  - list 6-2
  - loading 15-10
  - saving 15-10
  - updating through CLI 6-5
  - updating through xPC Target Explorer 6-3
  - updating through xPC Target Setup 15-8
- external mode
  - parameter tuning 3-40

## F

- file system objects
  - methods 16-7
  - xpctarget.fs introduction 8-4
- file systems
  - introduction 8-2
  - xpctarget.ftp 8-2
- Fortran
  - S- function wrapper 12-9
  - wrapper S-function 12-9
  - xPC Target 12-2
- FreeDOS
  - copying kernel 5-14
  - copying kernel/application 5-22
- FTP objects
  - xpctarget.ftp introduction 8-4
- functions
  - changing parameters 3-38
  - signal logging 3-30
  - signal monitoring 3-5
  - signal tracing 3-17

## G

- getting list of environment properties 6-2
- getting parameter properties 3-39
- getting signal properties 3-5

## H

- host scope viewer
  - xPC Target Explorer 3-16

## I

- interrupt mode
  - introduction 11-1

## K

- kernel
  - copying to flash memory 5-14
  - with DOSLoader mode 5-14
  - with StandAlone mode 5-21

## L

- list
  - environment properties 6-2
- loading
  - environment properties 15-10

## M

- MATLAB
  - parameter tuning 3-38
  - signal logging 3-30
  - signal monitoring 3-5
  - signal tracing 3-17
- methods
  - file system object 16-7

monitoring signals  
  xPC Target Explorer 3-2

## N

network communication  
  environment 15-4  
  specifying environment properties 15-4

## P

parameter tuning 3-40  
  overview 3-35  
  Web browser 3-42  
  with MATLAB 3-38  
  with remote control tool 15-26  
  with Simulink external mode 3-40  
  with xpcrcctool 15-26

parameters

  changing with commands 3-38  
  tuning with external mode 3-40  
  tuning with MATLAB 3-38  
  tuning with Web browser 3-42  
  tuning with xpcrcctool 15-26

polling mode

  introduction 11-1  
  setting up 11-6

properties

  changing environment 6-5  
  environment list 6-2  
  updating environment 6-5

## R

readxpcfile 8-10  
remote control tool  
  controlling target application 15-12  
  parameter tuning 15-26  
  signal logging 15-25  
  signal tracing 15-14

## S

saving environment properties 15-10

scope objects

  command-line interface 1-3  
  commands 1-3  
  creating 15-14  
  list of properties with files 3-22  
  list of properties with targets 3-19  
  methods, *see* commands  
  properties 1-3

scopes

  creating 3-8  
  stopping 3-15

selecting

  signals for tracing 15-14

serial communication

  environment 15-2

setting

  network communication 15-4  
  serial communication 15-2

setup dialog box

  C compiler 15-2

Setup window

  using 6-2

- signal logging
    - overview 3-28
    - with MATLAB 3-30
    - with remote control tool 15-25
    - with Web browser 3-34
  - signal monitoring
    - with MATLAB 3-5
  - signal tracing
    - overview 3-7
    - selecting signals 15-14
    - with MATLAB 3-17
    - with remote control tool 15-14
    - with Web browser 3-26
  - signals
    - adding 3-11
  - signals for tracing
    - selecting 15-14
  - Simulink external mode 3-40
    - parameter tuning 3-40
  - StandAlone mode 5-3
    - copying kernel/target application 5-22
    - creating kernel/application 5-21
    - updating environment properties 5-17
- T**
- target application
    - control target applications `xpcrcctool` 15-12
    - copying with StandAlone mode 5-22
    - remote control tool 15-12
    - with DOSLoader mode 5-16
  - target object properties
    - scopes of type file 3-21
  - target objects
    - changing parameters 3-38
    - command-line interface 1-2
    - commands 1-2
    - list of properties with files 3-21
    - methods, *see* commands
    - parameter properties 3-39
    - properties 1-2
    - signal properties 3-5
  - target PC
    - command-line interface 7-1
    - copying files with `xpctarget.ftp` 8-7
    - directory listings with `xpctarget.ftp` 8-5
    - disk information retrieval with `xpctarget.fs` 8-13
    - file content retrieval with `xpctarget.fs` 8-9
    - file conversion with `xpctarget.fs` 8-10
    - file information retrieval with `xpctarget.fs` 8-12
    - file removal with `xpctarget.fs` 8-11
    - file retrieval with `xpctarget.ftp` 8-6
    - list of open files with `xpctarget.fs` 8-11
    - manipulating scope object properties 7-6
    - manipulating scope objects 7-4
    - manipulating target object properties 7-3
    - using target application methods 7-2
  - Target PCs
    - installing, booting, and testing 4-8
  - task execution time (TET)
    - average 16-38
    - definition 3-33
    - in an example 3-34
    - logging 16-42
    - maximum 16-39
    - minimum 16-39
    - with the `getlog` function 16-45
  - TET, *see* task execution time

- tracing signals
    - xPC Target Explorer 3-7
  - troubleshooting
    - accessing documentation 13-5
    - advanced xPC Target 13-12
    - BIOS settings 13-4
    - boot disk 13-5
    - CAN boards 13-14
    - changed stop time 13-21
    - communication issues 13-3
    - connection lost 13-16
    - CPU Overload 13-13
    - custom device drivers 13-20
    - device drivers 13-20
    - different sample times 13-18
    - Error -10 13-20
    - file system disabled 13-22
    - general I/O 13-12
    - general xPC Target hints and tips 13-2
    - getxpcpci 13-4
    - host PC MATLAB halted 13-2
    - installation, configuration, and tests 13-6
    - invalid file ID 13-20
    - lost connection 13-16
    - models with CAN boards 13-14
    - new releases 13-5
    - PCI board slot and bus 13-15
    - PCI boards 13-4
    - sample time differences 13-18
    - sample times 13-18
    - stand-alone xPC Target application 13-21
    - stop time change 13-21
    - tagging virtual blocks 13-21
    - target PC halted 13-3
    - target PC monitor view 13-5
    - updated xPC Target releases 13-4
    - virtual block tagging 13-21
    - xPC Target PC unable to boot 13-2
    - xpctargetspy 13-5
    - xpctest 13-6
  - tuning parameters
    - xPC Target Explorer 3-35
- ## U
- updating environment properties through CLI 6-5
  - updating environment properties through xPC Target Explorer 6-3
  - updating environment properties through xPC Target Setup 15-8
  - user interface model 9-19
  - using setup window 6-2
  - using xPC Target setup window 6-2
- ## W
- Web browser
    - connecting 10-2
    - parameter tuning 3-42
    - signal logging 3-34
    - signal tracing 3-26
- ## X
- xPC Target
    - BIOS settings 13-4
    - custom GUIs 9-8
    - downloading and running applications 4-11
    - troubleshooting 13-1
    - user interface model 9-19
    - Web browser 10-1

- xPC Target Explorer
  - adding signals 3-11
  - configuring the host scope viewer 3-16
  - creating scopes 3-8
  - introduction 4-2
  - menu bar 4-17
  - monitoring signals 3-2
  - shortcut keys 4-17
  - stopping scopes 3-15
  - toolbar 4-17
  - tracing signals 3-7
  - tuning parameters 3-35
- xPC Target Setup window 6-2
- xpcrcctool
  - See also* remote control tool
  - signal tracing 15-14
- xpctarget.fs
  - creation 8-4
  - introduction 8-2
  - methods 16-8
  - overview 8-8
- xpctarget.fsbase
  - methods 16-7
- xpctarget.ftp
  - creation 8-4
  - introduction 8-2
  - methods 16-8
  - overview 8-5
- xpctcp2ser 10-5