xPC Target™
User's Guide

R2013a
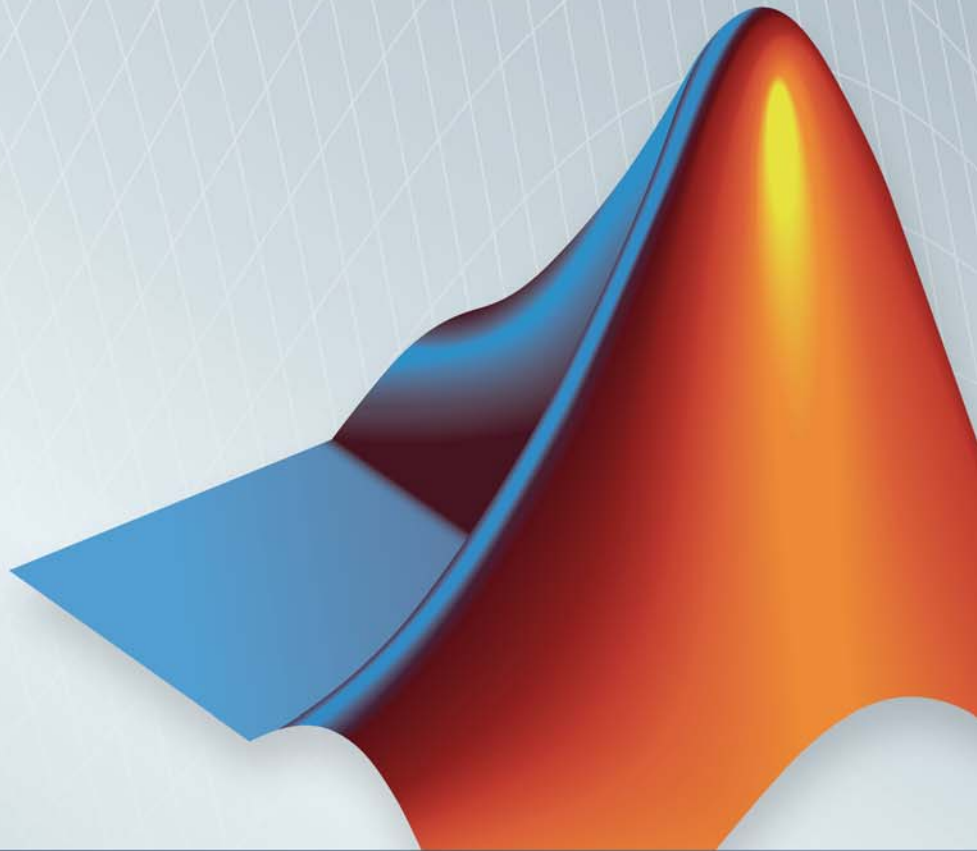
MATLAB®&SIMULINK®

MathWorks®

How to Contact MathWorks

	www.mathworks.com	Web
	comp.soft-sys.matlab	Newsgroup
	www.mathworks.com/contact_TS.html	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.
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 1999–2013 by The MathWorks, Inc.

Trademarks

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

Patents

MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

Revision History

Contents

Model Architectures

FPGA Models

1

<div style="text-align: right">

Vector CANape Support

</div>

2

<div style="text-align: right">

Incorporating Fortran S-Functions

</div>

3

Target Application Environment

4

Signals and Parameters

5

Execution Modes

6

Execution Using MATLAB® Scripts

Targets and Scopes in the MATLAB Interface

7

Logging Signal Data with FTP and File System Objects

8

Execution Using Graphical User Interface Models

9

Execution Using the Target Computer Command Line

10

Execution Using the Web Browser Interface

11

Troubleshooting

Confidence Test Failures

13

Host Computer Configuration

14

Target Computer Configuration

15

Application Parameters

22

Application Signals

23

Application Performance

24

Index

Model Architectures

xPC Target™ models are Simulink® models that use special blocks and architectures.

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

1

FPGA Models

FPGA Support

xPC Target and HDL Coder™ software enable you to implement Simulink algorithms and configure I/O functionality on Speedgoat field programmable gate array (FPGA) boards. Speedgoat I/O FPGA boards are sold as part of xPC Target Turnkey systems. For xPC Target Turnkey hardware, see `http://www.mathworks.com/products/xpctarget/supported-hardware/index.html`.

xPC Target supports the following Speedgoat boards.

Board	Description
Speedgoat IO301	Xilinx® Virtex-II, 6912 logic cells, 64 TTL I/O lines
Speedgoat IO302	Xilinx Virtex-II, 6912 logic cells, 32 RS-422 I/O lines
Speedgoat IO303	Xilinx Virtex-II, 6912 logic cells, 16 TTL and 24 RS-422 I/O lines
Speedgoat IO311	Xilinx Virtex-II, 24192 logic cells, 64 TTL I/O lines
Speedgoat IO312	Xilinx Virtex-II, 24192 logic cells, 32 RS-422 I/O lines
Speedgoat IO313	Xilinx Virtex-II, 24192 logic cells, 16 TTL and 24 RS-422 I/O lines
Speedgoat IO314	Xilinx Virtex-II, 24192 logic cells, 32 LVDS I/O lines
Speedgoat IO325	Xilinx Virtex-4 chip, 41472 logic cells, 64 LVCMOS or 32 LVDS (four are input only) I/O lines, two 16-bit 105 MHz analog input channels
Speedgoat IO331	Xilinx Spartan 6 chip, 147333 logic cells, 64 LVCMOS or 32 LVDS I/O lines

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

- Install HDL Coder and Xilinx ISE. For the specific ISE version required, see the Speedgoat board documentation. For more information, see "Tool Setup" in the HDL Coder documentation.

- Install the Speedgoat FPGA I/O board in the target computer.

- Be familiar with FPGA technology. In particular, you must know the clock frequency and the I/O connector pin and channel configuration of your FPGA board.

- Have experience using data type conversion and designing Simulink fixed-point algorithms.

To generate HDL code for your FPGA target, you do not need to have HDL programming experience.

The xPC Target product provides the following FPGA applications as examples.

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

FPGA Programming and Configuration

To implement Simulink algorithms on a Speedgoat FPGA I/O board, you use "HDL Workflow Advisor" to specify an FPGA board and its I/O interface, synthesize the Simulink algorithm for FPGA programming, and generate an xPC Target interface subsystem model. The interface subsystem model contains blocks to program the FPGA and communicate with the FPGA I/O board during target application execution. You add the generated subsystem to your xPC Target domain model.

The workflow looks like this figure.
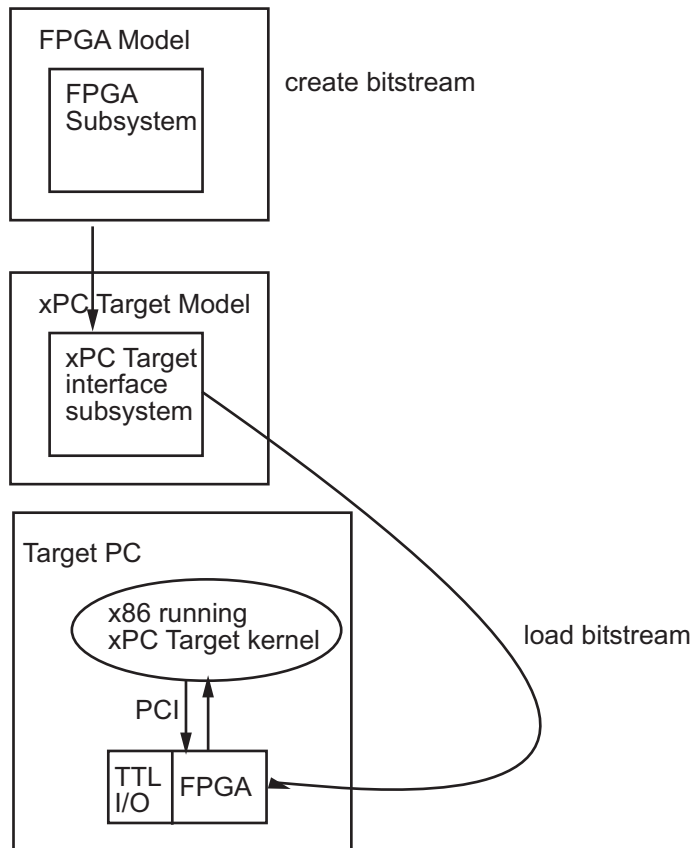
Before you begin this procedure, you must have completed the following:

- "Simulink Domain Model" on page 1-6
- "FPGA Subsystem Plan" on page 1-8

This procedure uses example `Servo Control with the Speedgoat IO301 FPGA Board`.

1 "FPGA Target Configuration" on page 1-11

2 "FPGA Target Interface Configuration" on page 1-13

3 "FPGA Target Frequency Configuration" on page 1-15

4 "xPC Target Interface Subsystem Generation" on page 1-17

5 "xPC Target Domain Model" on page 1-21

6 "xPC Target Interface Subsystem Integration" on page 1-23

The next task is "Target Application Execution" on page 1-25.

Simulink Domain Model

The Simulink FPGA domain model contains a subsystem (algorithm) to be programmed onto the FPGA chip. Using this model, you can test your FPGA algorithm in a simulation environment before you deploy the algorithm to an FPGA board.

1 Create a Simulink model to contain the algorithm that you want to load onto the FPGA.

2 Place the algorithm to be programmed on the FPGA inside a Subsystem block. The model can include other blocks and subsystems for testing. However, one subsystem must contain the FPGA algorithm.

3 Set or confirm the subsystem inport and outport names and data types.

   The HDL Workflow Advisor uses these settings for routing and mapping algorithm signals to I/O connector channels. See "FPGA Subsystem Plan" on page 1-8.

4 Save the model.

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

For an example of an FPGA domain model, see `dxpcSGIO301servo_fpga`. The ServoSystem subsystem contains the FPGA algorithm.
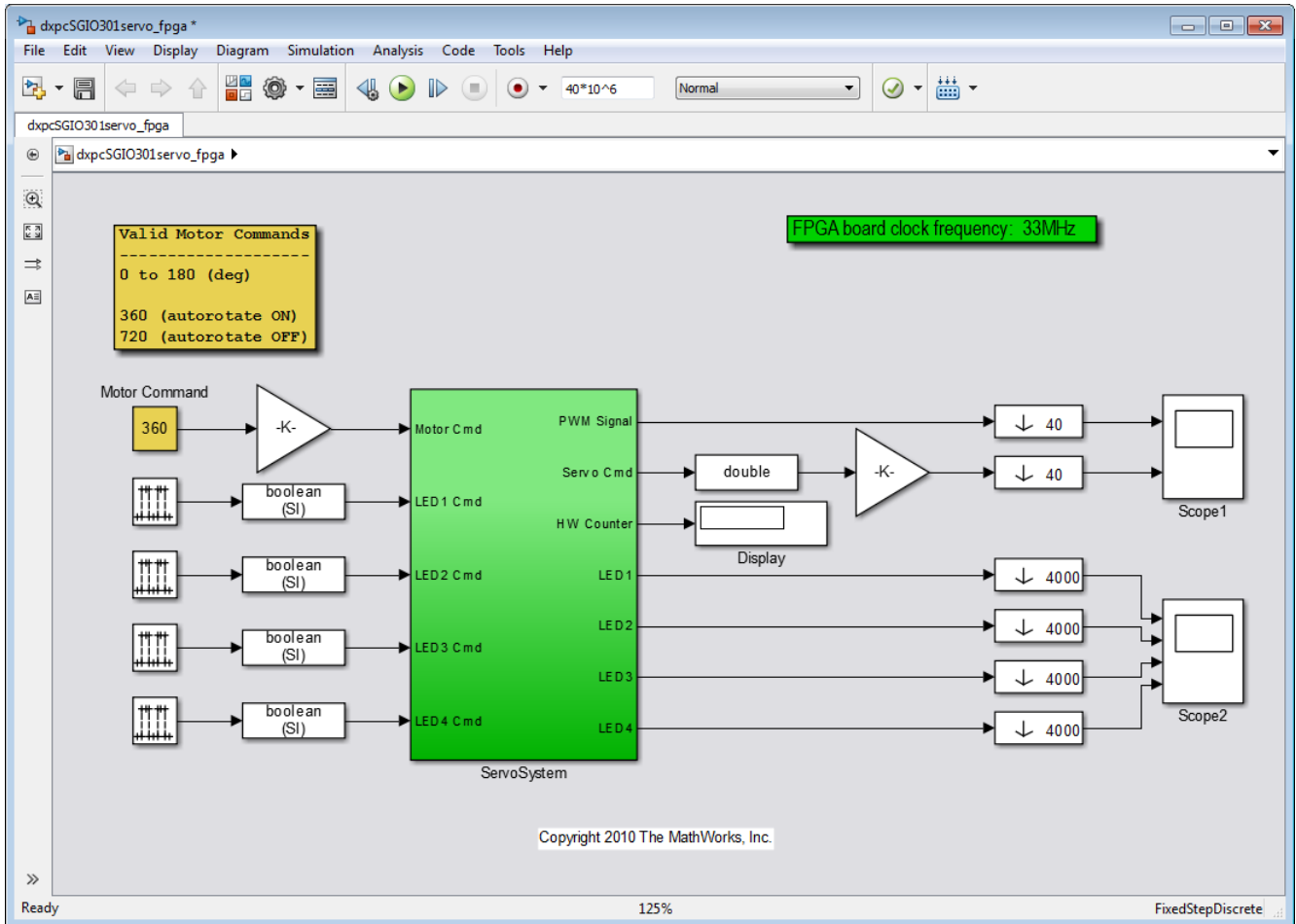
FPGA Subsystem Plan

Before you start working with the HDL Workflow Advisor, you must plan how to prepare the FPGA subsystem for HDL code generation and FPGA synthesis.

Target Device

You must decide which FPGA to target for code generation. The example procedure uses the FPGA Turnkey target workflow and the Speedgoat IO301 FPGA IO board for target platform. These choices require that you use the Xilinx ISE synthesis tool.

For information about other target devices, see "Supported Third-Party Tools and Hardware".

FPGA Synchronization Mode

To select the processor/FPGA synchronization mode, you must decide which of the FPGA synchronization modes to use:

- `Free running`
- `Coprocessing   blocking`
- `Coprocessing   nonblocking with delay`.

For more information, see "FPGA Synchronization Modes" on page 1-29.

FPGA Inports and Outports

Inports and outports may transmit signal data between the target computer and the FPGA over the PCI bus or map to I/O channels for communicating with external devices. For connector pin and I/O channel assignments of your supported FPGA I/O board, see the board reference page for your board.

In addition to the **Port Name** and **Port Type** (`Inport` or `Outport`), to specify the I/O interface, see:

- **Data Type**—Encodes such attributes as width and sign. Data types must map consistently to their corresponding I/O pins. An inport of type `Boolean` requires 1 bit, one of type `uint32` requires 32 bits, and so on. For example,

you cannot connect an inport of type `uint32` to an FPGA I/O interface of type `TTL I/O channel [0:7]`; it requires `TTL I/O channel [0:31]`.

- **Target Platform Interfaces**—Encodes the I/O channels on the FPGA as well as their functional type. For a single-ended interface (`TTL`, `LVCMOS`), one channel maps to one connector pin. For a differential interface (`RS422`, `LVDS`), one channel maps to two connector pins. To discover the mapping for a particular pin, see the pin connector map provided with the board description.

  I/O channels may also map to a predefined specification or role (`PCI Interface`, `Interrupt from FPGA`).

  For information on using FPGA interrupts, see "Interrupt Configuration" on page 1-26.

- **Bit Range/Address/FPGA Pin**—Encodes the pins on the target platform to which the inports and outports are assigned, along with the channel number used by the port. For specification `PCI Interface`, **Bit Range/Address/FPGA Pin** encodes the PCI address used by the port.

If vector inports or outports are required, specify a vector port:

- `Inport` — Add a mux outside the subsystem that connects to a demux inside the subsystem.

- `Outport` – Add a mux inside the subsystem that connects to a demux outside the subsystem.

- `Inport` and `Outport` – Configure the port dimension to be greater than 1.

Workflow Advisor automatically inserts a strobe to achieve a simultaneous update of vector elements.

If you have specified vector inports or outports, before generating code, you must select the **Scalarize vector ports** check box. This check box is on the **Coding style** tab of node **Global Settings**, under node **HDL Code Generation** in the Configuration Parameters dialog box.

FPGA Clock Frequency

The FPGA system clock frequency defaults to the fixed FPGA input clock frequency shown in the **FPGA input clock frequency (MHz)** box. You can

specify another frequency in the **FPGA system clock frequency (MHz)** box. If the FPGA cannot generate the specified value exactly, HDL Workflow Advisor generates the closest match, F_{system}, based on the following formula:

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

F_{input} is the fixed FPGA input clock frequency. `ClkFxMultiply` and `ClkFxDivide` are integers.

FPGA Target Configuration

This procedure uses the dxpcSGIO301servo_fpga example. You must have already created an FPGA subsystem (algorithm) in an FPGA domain model and developed an FPGA subsystem plan. See "Simulink Domain Model" on page 1-6 and "FPGA Subsystem Plan" on page 1-8.

1 Open the FPGA domain model dxpcSGIO301servo_fpga.

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

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

3 Expand the **Set Target** folder and select task **1.1 Set Target Device and Synthesis Tool**.

4 Set **Target Workflow** to FPGA Turnkey.

5 From the **Target platform** drop-down list, select the Speedgoat FPGA I/O board installed in your target computer.

For the dxpcSGIO301servo_fpga example, this is Speedgoat IO301 FPGA IO board (Acromag PMC-DX501).

6 Click **Run This Task**.
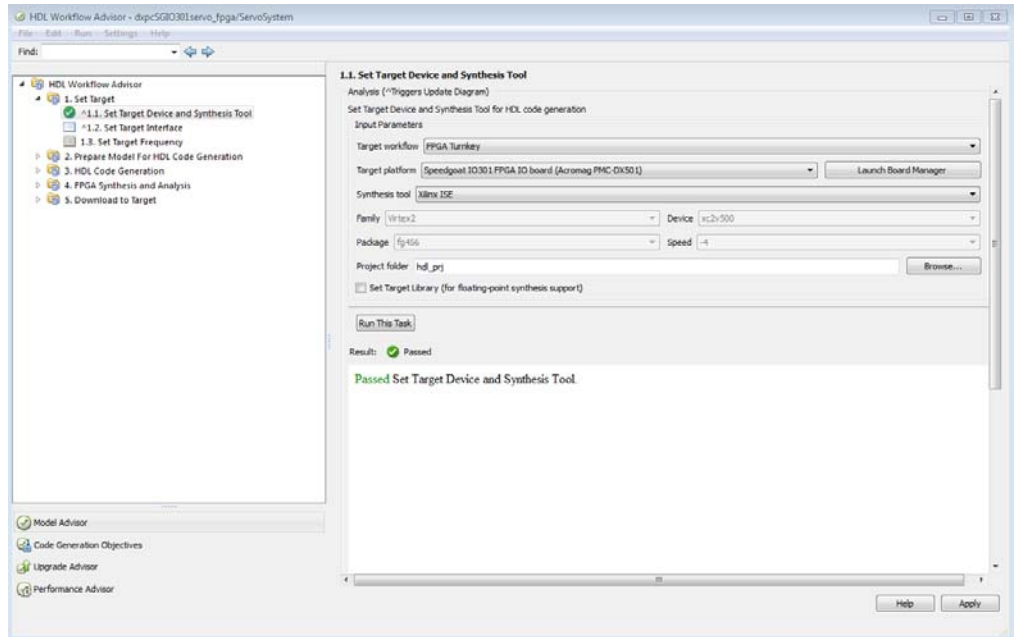
The next task is "FPGA Target Interface Configuration" on page 1-13.

FPGA Target Interface Configuration

This procedure uses the dxpcSGIO301servo_fpga example. You must have already developed an FPGA subsystem plan and configured the FPGA target. See "FPGA Subsystem Plan" on page 1-8 and "FPGA Target Configuration" on page 1-11.

1 In the **Set Target** folder, select task **1.2 Set Target Interface**.

2 In the **Processor/FPGA synchronization** box, select Free running.

For information about FPGA synchronization modes, see "FPGA Synchronization Modes" on page 1-29.

3 For signals from the FPGA through I/O lines (channels) — In the **Target Platform Interfaces** column, select the required I/O channel type (for example, TTL I/O Channel [0:63]).

In the **Bit Range/Address/FPGA Pin** column, enter the channel value for each signal.

4 For signals between the target computer and the FPGA — In the **Target Platform Interfaces** column, select PCI Interface.

In the **Bit Range/Address/FPGA Pin** column, use the automatically generated values. Do not enter PCI address values.

5 After specifying interfaces for the required signals, click **Run This Task**.
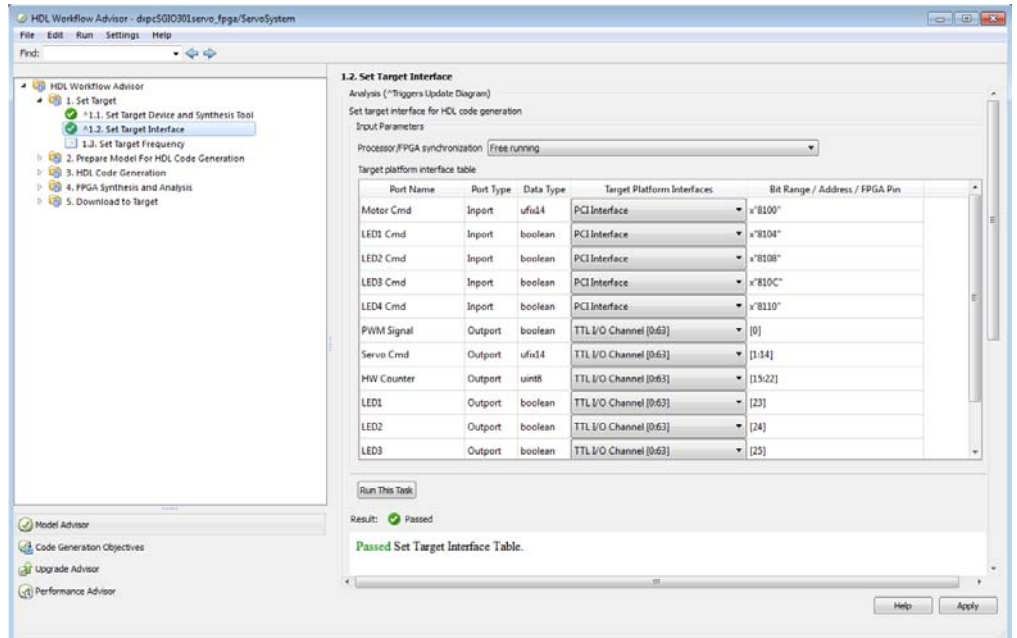
For more information on mapping Speedgoat FPGA I/O pins in HDL Workflow Advisor, see "Set the Target Interface for Speedgoat Boards".

The next task is "FPGA Target Frequency Configuration" on page 1-15.

FPGA Target Frequency Configuration

This optional procedure uses the `dxpcSGIO301servo_fpga` example. You must have already developed an FPGA subsystem plan and configured the FPGA target interface. See "FPGA Subsystem Plan" on page 1-8 and "FPGA Target Interface Configuration" on page 1-13.

1 In the **Set Target** folder, select task **1.3 Set Target Frequency** (optional).

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

2 To specify a different system clock frequency (for example, 50 MHz), type the new value in the field **FPGA system clock frequency (MHz)**. For the permitted range for the system clock rate, see the Speedgoat board characteristics table.

The system may set a value different from the one you specified. For more information, see "FPGA Clock Frequency" on page 1-9.

3 Click **Run This Task**.
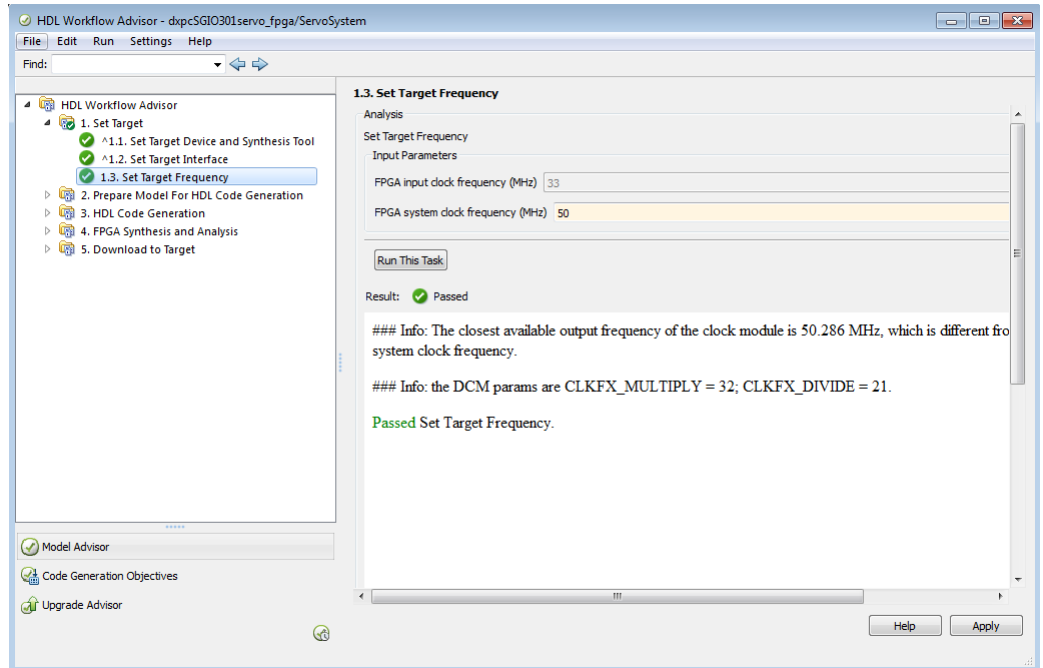
The next task is "xPC Target Interface Subsystem Generation" on page 1-17.

xPC Target Interface Subsystem Generation

This procedure uses the `dxpcSGIO301servo_fpga` example. You must have already configured the FPGA target interface and the required target frequency. If you have specified vector inports or outports, you must have already selected the **Scalarize vector ports** check box. This check box is on the **Coding style** tab of node **Global Settings**, under node **HDL Code Generation** in the Configuration Parameters dialog box.

1 Expand the **Download to Target** folder, and right-click task **5.2 Generate xPC Target Interface**.

2 In this pane, click **Run To Selected Task**.

This action:

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

HDL Workflow Advisor looks like this figure.
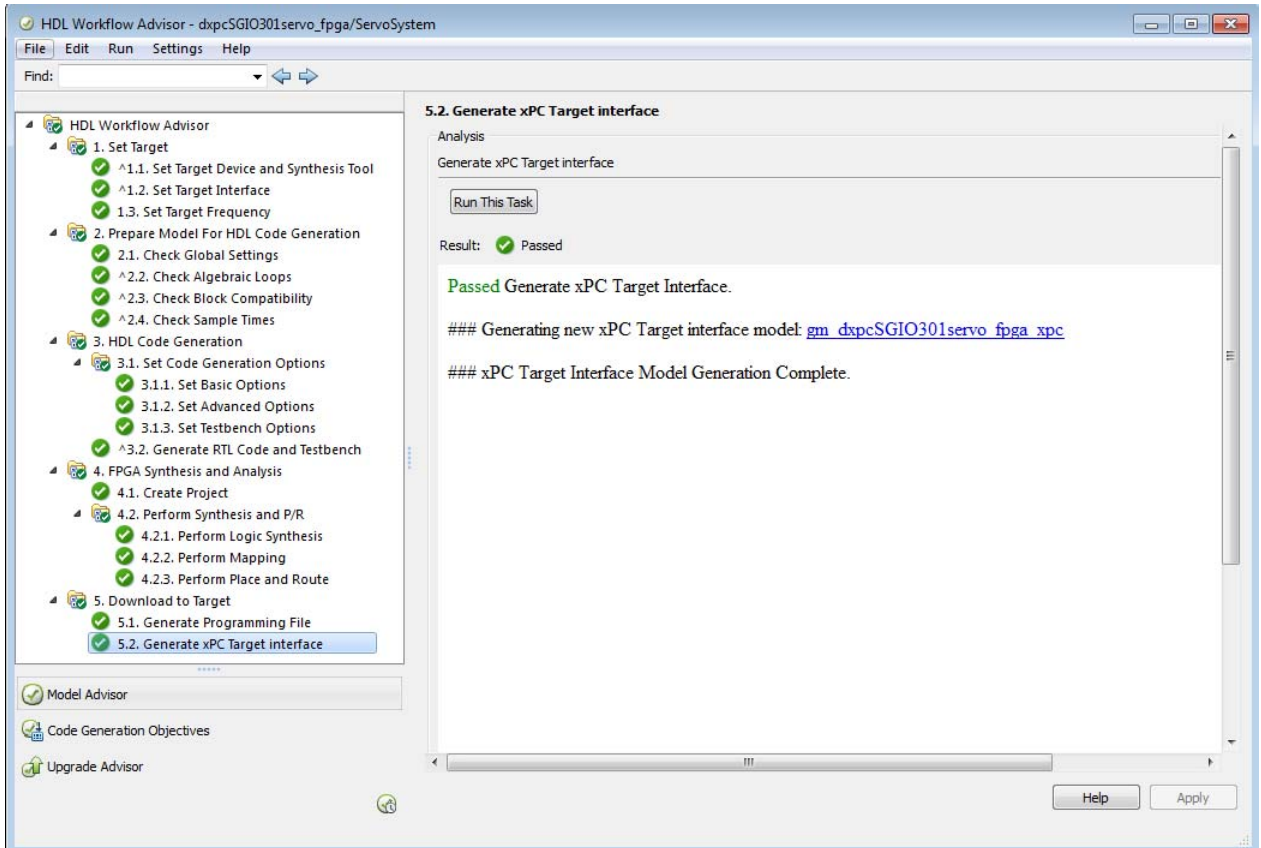
The generated interface subsystem looks like this figure.
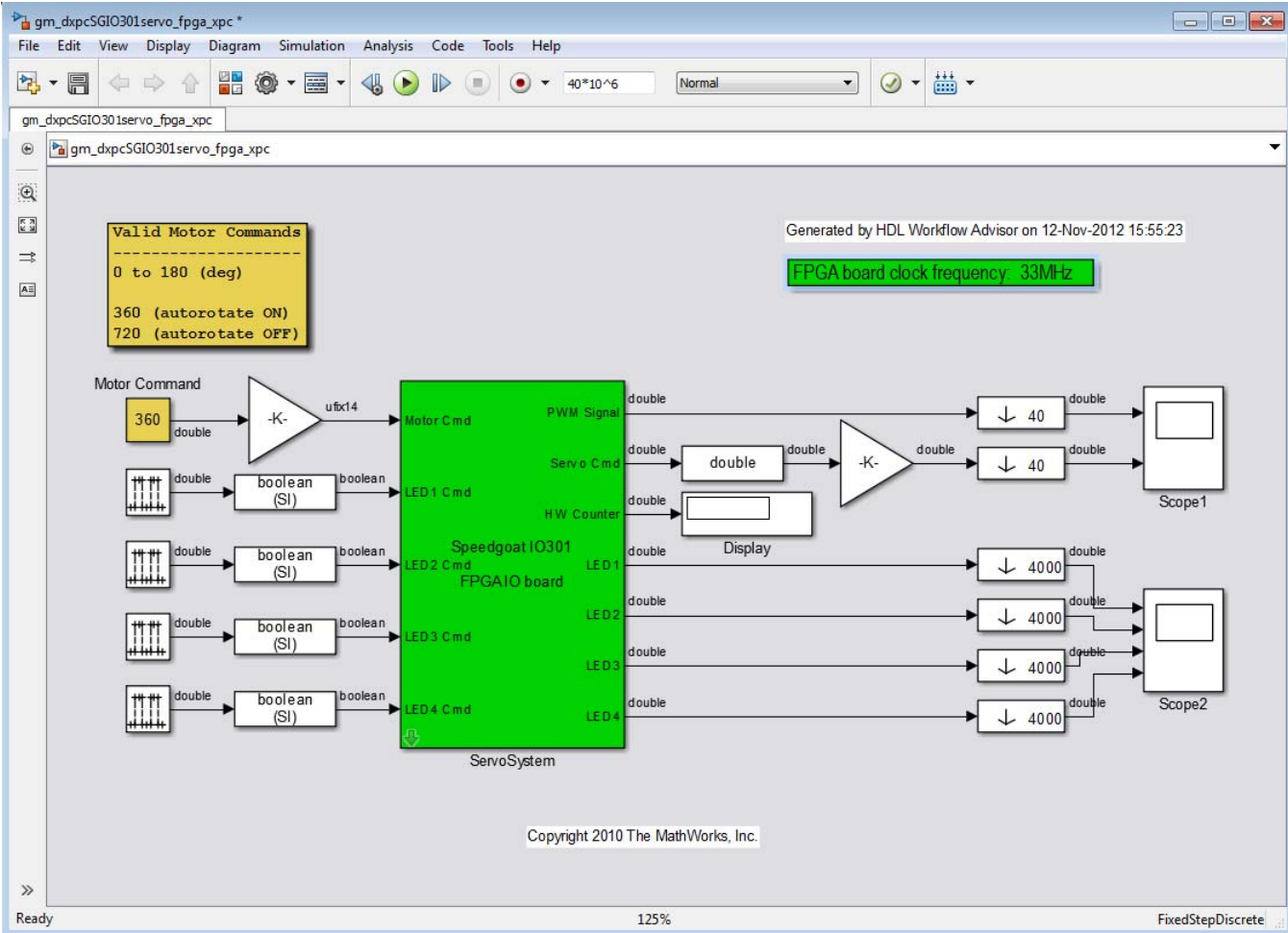
This generated model contains a masked subsystem with the same name as the subsystem in the Simulink FPGA domain model. Although the appearance is similar, this subsystem does not contain the Simulink algorithm. Instead, the algorithm is implemented in an FPGA bitstream. You reference and load this algorithm into the FPGA from this subsystem.

The next task is "xPC Target Domain Model" on page 1-21.

xPC Target Domain Model

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

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

2 Save the model.

In example model `dxpcSGIO301servo_xpc`, the disconnected signals connect to the inports and outports of the xPC Target interface subsystem.
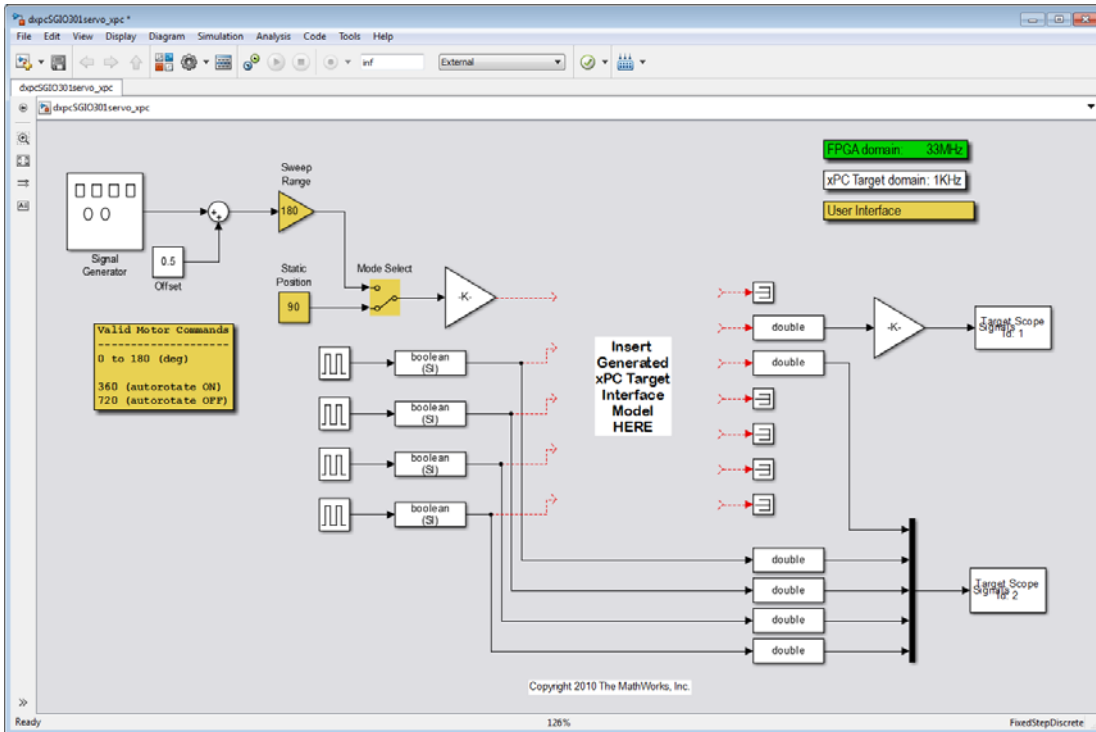
The next task is "xPC Target Interface Subsystem Integration" on page 1-23.

xPC Target Interface Subsystem Integration

Before doing this procedure, you must have already generated an xPC Target interface subsystem with the HDL Coder software. If you have not yet done so, see "xPC Target Interface Subsystem Generation" on page 1-17.

You need to set three parameters in the xPC Target interface subsystem mask:

- Device index
- PCI slot
- Sample time

In addition, you must evaluate the communication timeout requirements for your model. The default communication timeout of 5 seconds may not be long enough to download and program a large FPGA, such as the IO331.

1 In the Simulink editor, open gm_*fpgamodelname*_xpc.

2 Copy and paste the this subsystem, xPC Target interface subsystem, into the xPC Target domain model.

3 Save or discard gm_*fpgamodelname*_xpc. You can recreate it as required using the HDL Workflow Advisor.

4 In the xPC Target domain model, connect signals to the inports and outports of the xPC Target interface subsystem.

5 Set the block parameters according to the FPGA I/O boards in your target computer.

- If you have a single FPGA I/O board, leave the device index and PCI slot at the default values. You can set the sample time or leave it at −1 for inheritance.

- If you have multiple FPGA I/O boards, give each board a unique device index.

- If you have two or more boards of the same type (for example, two IO301s), specify the PCI slot ([bus, slot]) for each board. Get this information with the xpctarget.xpc.getxpcpci function.

6 If you need a larger communication timeout, in the Configuration Parameters dialog box for the model, expand the **Code Generation** and **xPC Target options** nodes, clear the **Use default communication timeout check** box, and then enter a new value in the **Specify the communication timeout in seconds** box.

7 Save the model.

For an example of an xPC Target domain model that has the interface subsystem pasted and connected, see dxpcSGIO301servo_xpc_wiss.
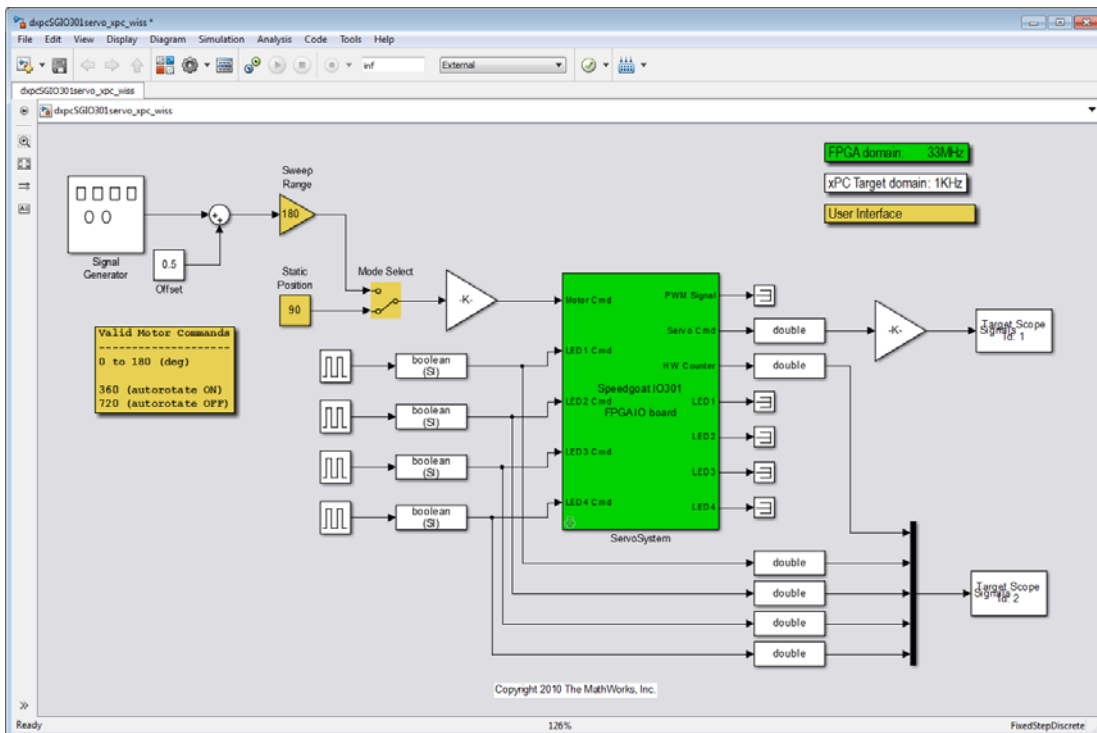
You are now ready to build and download the xPC Target domain model. Continue with "Target Application Execution" on page 1-25.

Target Application Execution

To do this procedure, you have already created an xPC Target domain model that includes an xPC Target interface subsystem generated from the HDL Workflow Advisor. If you have not yet done so, see "xPC Target Interface Subsystem Integration" on page 1-23.

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

2 Build and download the xPC Target model. The xPC Target model loads onto the target computer and the FPGA algorithm bitstream loads onto the FPGA.

3 If you are using I/O lines (channels), confirm that you have connected the lines to your external hardware under test.

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

Interrupt Configuration

xPC Target software schedules the target application using either the internal timer of the target computer (default) or an interrupt from an I/O board. You can use your Speedgoat FPGA board to generate an interrupt, which allows you to:
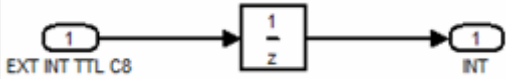
- Schedule execution of the target application based on this interrupt (synchronous execution). For this method, you must generate the interrupt periodically.

- Execute a designated subsystem in your target application (asynchronous execution).

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

FPGA Domain Model

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

Source	Description
Internal	A clock you create using Simulink blocks to create input signals. This clock is a binary pulse train of zeros and ones (transition from 0 to 1 and vice versa). The clock generates an interrupt on a rising edge. The following is an example of an internally generated interrupt source from Simulink blocks. Connect the internally generated interrupt source to an outport labelled INT.
External	A clock signal that comes from a device outside the target computer. You use a digital input pin to connect to this signal. The following is an example of an externally generated

Source	Description
	interrupt source that comes from TTL channel 8. Delay this source by one FPGA clock cycle and connect to an outport labeled INT.

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

You are now ready to set up interrupt support in the xPC Target domain model. See "xPC Target Domain Model" on page 1-27.

xPC Target Domain Model

If you have not yet done so, for overview information, see "FPGA Domain Model" on page 1-26.

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

1 Open the xPC Target domain model.

2 In the Simulink editor, select **Simulation > Model Configuration Parameters**.

3 Navigate to node **xPC Target options**, under node **Code Generation**.

4 From the **Real-time interrupt source** list, select one of the following:

- `Auto (PCI only)`

- The IRQ assigned to your FPGA board

5 From the **I/O board generating the interrupt** parameter, select your FPGA board, for example, `Speedgoat_IO301`.

6 Add the xPC Target interface subsystem to the model (see "xPC Target Interface Subsystem Integration" on page 1-23).

7 Build and download the application to the target computer.

8 When you start the target application, simulation updates occur when the application receives an interrupt from the FPGA I/O board.

FPGA Synchronization Modes

In xPC Target, an FPGA operates in three synchronization modes:

- `Free running`
- `Coprocessing   blocking`
- `Coprocessing   nonblocking with delay`

- `Free running` (default) — The CPU and the FPGA each run nonsynchronized, continuously, and in parallel. When you want the CPU to run continuously without interrupts, select this mode. For example, you could select this mode when the model is processing continuous PWM output.

  The target computer CPU strobes data out of the FPGA, reads the results from the FPGA outputs, writes data to the FPGA inputs, and strobes the data into the FPGA.
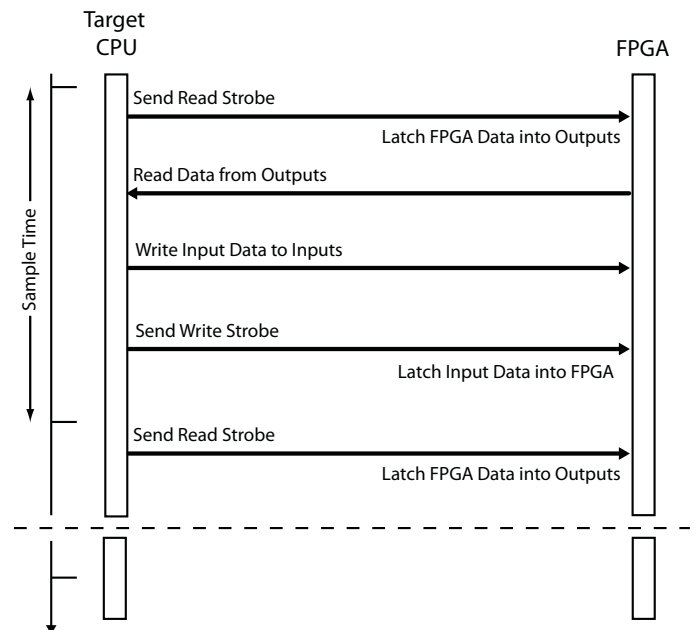
- `Coprocessing    blocking` — The CPU and the FPGA run synchronized and in tandem. When the FPGA execution time is short compared to the target computer sample time, and you want the FPGA to complete before the model continues, select this coprocessor mode.

  The CPU writes data to the FPGA inputs, strobes the data into the FPGA, waits for the FPGA to finish executing, and reads the result out of the FPGA outputs.
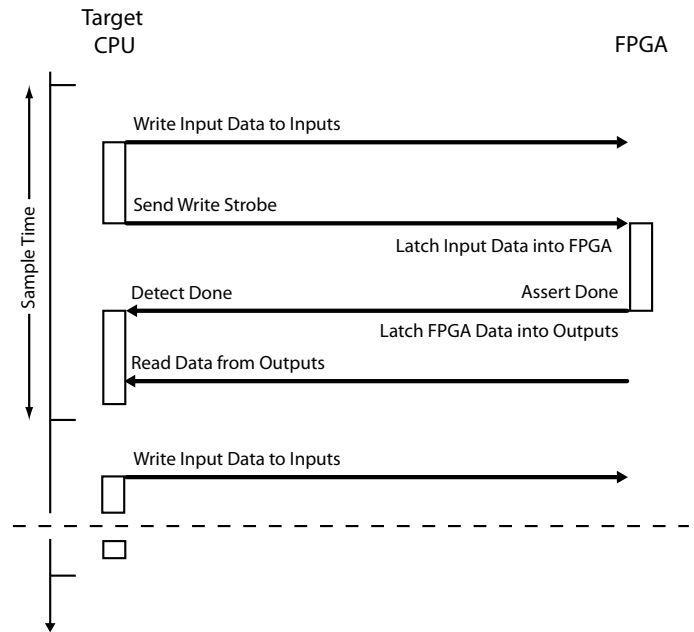
- `Coprocessing    nonblocking with delay` — The CPU and the FPGA run synchronized and in tandem. When the FPGA execution time is long compared to the target computer sample time, select this coprocessor mode. For example, you could select this mode to manage multiple FPGAs effectively in parallel.

  The CPU waits for the FPGA to finish executing, reads the data from the previous time step, writes new data to the FPGA inputs, and strobes the data into the FPGA.

Target
CPU   FPGA

Detect Done   Assert Done

Latch FPGA Data into Outputs

Read Data from Outputs

Write Input Data to Inputs

Send Write Strobe

Latch Input Data into FPGA

Detect Done   Assert Done

Latch FPGA Data into Outputs

Sample Time

1-31
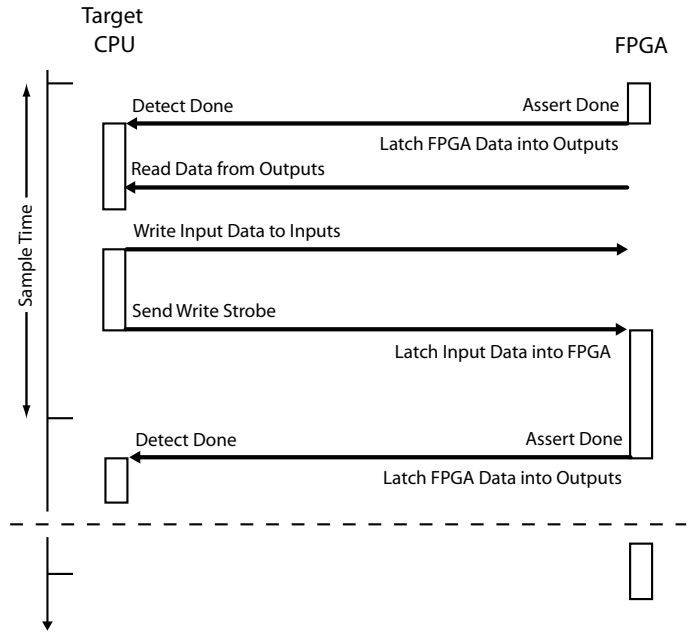
Vector CANape Support

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

Vector CANape

Vector CANape Basics

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

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

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

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

• Configure Vector CANape to communicate with the xPC Target software as an ECU.

• Enable the xPC Target software to generate a target application that can provide data compliant with Vector CANape.

• Provide a standard TCP/IP physical layer between the host computer and target computer. The xPC Target software supports Vector CANape only through TCP/IP.

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

- An XCP server process in the target application that runs on-demand in the background.

- A generator that produces A2L (ASAP2) files that Vector CANape can load into the Vector CANape software database. The generated file contains signal and parameter access information for the target application.

xPC Target and Vector CANape Limitations

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

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

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

- Vector CANape calibration commands or flash RAM calibration commands

- Multiple simultaneous Vector CANape connections to a single target computer

Configuring the Model for Vector CANape

In this section...
"Setting Up and Building the Model" on page 2-4
"Creating a New Vector CANape Project" on page 2-5
"Configuring the Vector CANape Device" on page 2-6
"Providing A2L (ASAP2) Files for Vector CANape" on page 2-9

Setting Up and Building the Model

Set up your model to work with Vector CANape. The following procedure uses the xpcosc model. It assumes that you have already configured your model to generate xPC Target code. If you have not done so, see "Set Configuration Parameters" and "xPC Target Options Configuration Parameters" on page 4-3. It also assumes that you have already created a Vector CANape project. If you have not done so, see "Creating a New Vector CANape Project" on page 2-5.

1 In the MATLAB® Command Window, type

   xpcosc

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

   xpclib

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

4 Drag the XCP Server block to the xpcosc model.

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

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

   • **Target Address** — Target IP address for target computer. The default value is getxpcenv(`TcpIpTargetAddress'). Typically, you will want to leave the default entry. Otherwise, enter the TCP/IP address for the target computer.

> • **Server Port** — Port for communication between target computer and XCP server. The default value is 5555. This value must be the same as the port number you specify for the Vector MATLAB device.

6 If you want to use the event mode to acquire signal data, set the priority of the xcpserver block to be the lowest priority. For example, enter a priority of 10000000. For Simulink blocks, the higher the priority number, the lower the priority.

7 In the model Simulink window, click **Simulation > Model Configuration Parameters**.

The Configuration Parameters dialog box is displayed for the model.

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

The associated pane is displayed.

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

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

10 Build the model.

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

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

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

You can now create a new Vector CANape project (see "Creating a New Vector CANape Project" on page 2-5.

Creating a New Vector CANape Project

This procedure describes how to create a new Vector CANape project that can communicate with an xPC Target application. It assumes that you have

set up, built, and downloaded your model (see "Setting Up and Building the Model" on page 2-4).

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

```
mkdir C:\MyProject
```

2 Start Vector CANape.

3 Select **File > New project**.

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

4 After you create the new project, start it.

After the preliminary warning, the CANape window is displayed.

You can now configure the target computer and the loaded target application as a Vector CANape device (see "Configuring the Model for Vector CANape" on page 2-4).

Configuring the Vector CANape Device

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

- You have created a new Vector CANape project to associate with a particular target application. If you have not yet done so, see "Creating a New Vector CANape Project" on page 2-5.

- You have set up, built, and downloaded your model. If you have not yet done so, see "Setting Up and Building the Model" on page 2-4.

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

The Vector CANape window is displayed.

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

The device configuration window is displayed.

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

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

```
xPCTarget
```

Add the required comments.

5 Click **Next**.

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

7 Click **Driver settings**.

The XCP driver settings window is displayed.

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

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

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

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

11 Set the port number to 5555.

12 Click **OK**.

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

  a In the **Driver** pane of the XCP driver settings window, click **Extended driver settings**.

  b Set the **ODT_ENTRY_ADDRESS_OPT_DISABLED** parameter to Yes.

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

  c Click **OK**.

14 In the XCP driver settings window, verify the connection to the target computer by clicking **Test connection**. This command succeeds only if the target computer is running and connected to exactly one host computer.

15 Click **OK**.

The **Device** dialog is displayed.

16 Click **Next**.

Do not exit the dialog.

You can now configure the location of the target application A2L (ASAP2) file for the CANape database. See "Configuring the Location of the A2L (ASAP2) File" on page 2-8.

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

Configuring the Location of the A2L (ASAP2) File

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

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

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

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

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

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

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

A dialog requests confirmation of ASAP2 settings.

5 Click **Yes**.

6 Click **Next**.

7 Click **Next**.

8 Click **Next**.

9 Click **OK**.

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

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

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

Providing A2L (ASAP2) Files for Vector CANape

This topic assumes that:

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

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

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

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

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

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

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

Event Mode Data Acquisition

In this section...
"Guidelines" on page 2-11
"Limitations" on page 2-11

Guidelines

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

- Set the priority of the `xcpserver` block to the lowest possible. See suggested priority values in "Setting Up and Building the Model" on page 2-4.

- The xPC Target software generates events at the base sample rate; this execution rate is the fastest possible. If you are tracing a signal that is updated at a slower rate than the base sample rate, you must decimate the data to match the actual execution. (The xPC Target software generates the event name with the ASAP2 generation during model code generation.)

- You can associate signals with the event generation through the Vector CANape graphical user interface.

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

Limitations

The event mode data acquisition has the following limitations:

- Every piece of data that the xPC Target software adds to the event list slows down the target application. The amount of data that you can observe depends on the model sample time and the speed of the target computer. It is possible to overload the target computer CPU to the point where data integrity is reduced.

- You can only trace signals and scalar parameters. You cannot trace vector parameters.

Incorporating Fortran S-Functions

Fortran S-Functions

The xPC Target product supports Fortran in Simulink models using
S-functions. For more details, see "Create Level-2 Fortran S-Functions" and
"Port Legacy Code".

In this section...

Prerequisites

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

http://www.mathworks.com/support/compilers/current_release/

Simulink Demos Folder

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

1 In the MATLAB Command Window, type

   demos

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

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

   A list of Simulink examples appears.

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

The associated Simulink examples page opens.

4 Click **Open this model**.

S-function examples are displayed.

5 Double-click the Fortran S-functions block.

Fortran S-functions and associated templates appear.

Steps to Incorporate Fortran

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

1 Using the Fortran compiler, compile the Fortran code (subroutines (`*.f`)). You will need to specify particular compiler options.

2 Write a Simulink C-MEX wrapper S-function. This wrapper S-function calls one or more of the Fortran subroutines in the compiled Fortran object code from step 1.

3 Use the `mex` function to compile this C-MEX S-function using a Visual C/C++ compiler. Define several Fortran run-time libraries to be linked in.

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

4 Run a simulation C-MEX file with the Simulink software to validate the compiled Fortran code and wrapper S-function.

5 Copy relevant Fortran run-time libraries to the application build folder for the xPC Target application build.

6 Define the Fortran libraries, and the Fortran object files from step 1, in the Simulink Coder™ dialog box of the Simulink model. You must define these libraries and files as additional components to be linked in when the xPC Target application link stage takes place.

7 Initiate the xPC Target specific Simulink Coder build procedure for the example model. Simulink Coder builds and downloads xPC Target onto the target computer.

Fortran Atmosphere Model

This example uses the example Atmosphere model that comes with the Simulink product. The following procedures require you to know how to write Fortran code according to Simulink and xPC Target software requirements.

Before you start, create an xPC Target Simulink model for the Atmosphere model. See "Creating a Fortran Atmosphere Model" on page 3-4.

Creating a Fortran Atmosphere Model

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

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

2 Type

sfcndemo_atmos

The sfcndemo_atmos model is displayed.

3 Add an xPC Target Scope block of type Target.

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

The model sfcndemo_atmos should look like the figure shown.
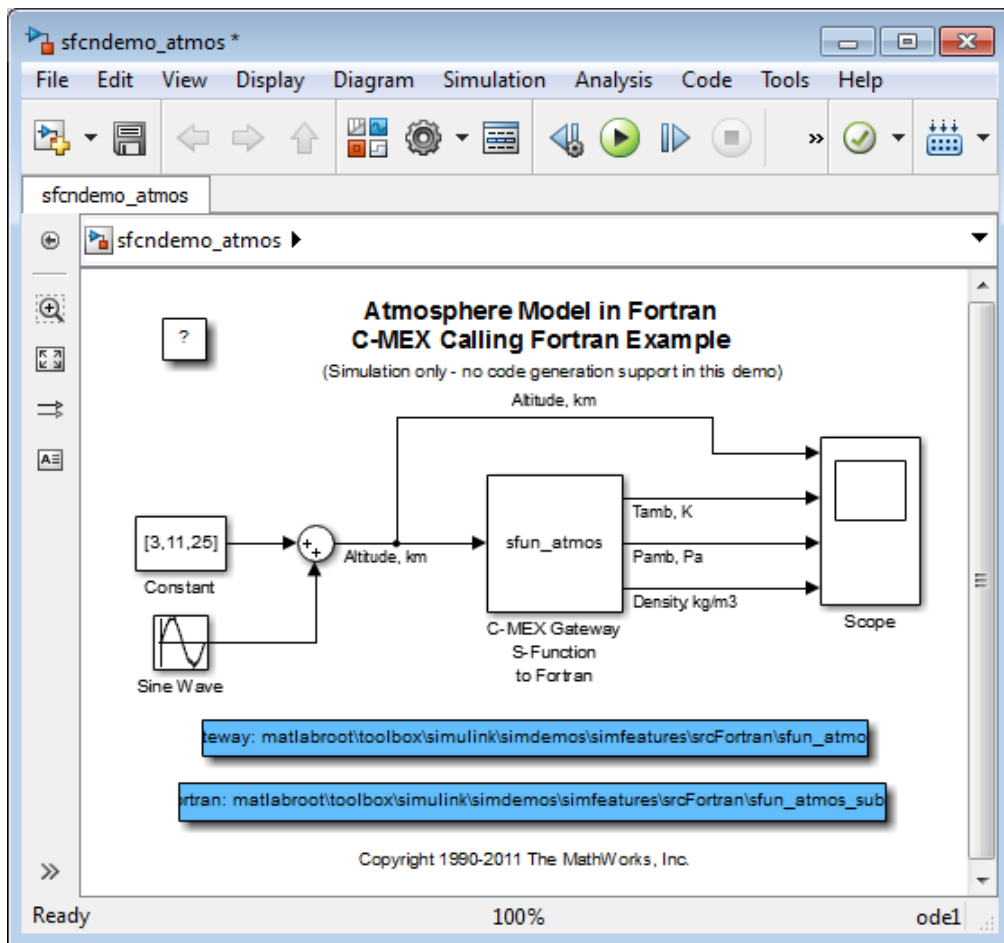
5 Double-click the target Scope block.

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

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

8 Click **Apply**, then **OK**.

9 Double-click the Sine Wave block.

10 For the **Sample time** parameter, enter 0.05.

11 Click **OK**.

12 From the **File** menu, click **Save as**. Browse to your current working folder, for example, xpc_fortran_test. Enter a filename. For example, enter fortran_atmos_xpc and then click **Save**.

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

Compiling Fortran Files

1 In the MATLAB Command Window, copy the file sfun_atmos_sub.F into your Fortran working folder, for example, xpc_fortran_test. This is sample Fortran code that implements a subroutine for the Atmosphere model.

2 From *Fortran_compiler_dir*\lib\ia32, copy the following files to the working folder:

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

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

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

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

Creating a C-MEX Wrapper S-Function

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

• Compile your Fortran code. See "Compiling Fortran Files" on page 3-7.

• Become familiar with the guidelines and calling conventions for Simulink Fortran level 2 S-functions (see "Create Level-2 Fortran S-Functions").

• Implement the required callback functions using standard functions to access the fields of the S-function's simulation data structure, SimStruct (see "Templates for C S-Functions").

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

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

---

1 Copy the file sfuntmpl_gate_fortran.c to your working folder.

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

2 With a text editor of your choice, open sfuntmpl_gate_fortran.c.

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

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

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

```
#define S_FUNCTION_NAME sfun_atmos
```

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

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

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

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

with

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

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

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

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

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

#define TO(S) (ssGetSFcnParam(S, TO_IDX))
#define PO(S) (ssGetSFcnParam(S, PO_IDX))
#define RO(S) (ssGetSFcnParam(S, RO_IDX))
```

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

```
static void mdlInitializeSizes(SimStruct *S)
```

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

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

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

  ssSetNumContStates( S, 0 );
  ssSetNumDiscStates( S, 0 );
  ssSetNumInputPorts(S, 1);
  ssSetInputPortWidth(S, 0, 3);
  ssSetInputPortDirectFeedThrough(S, 0, 1);
  ssSetInputPortRequiredContiguous(S, 0, 1);
  ssSetNumOutputPorts(S, 3);
  ssSetOutputPortWidth(S, 0, 3); /* temperature */
  ssSetOutputPortWidth(S, 1, 3); /* pressure    */
  ssSetOutputPortWidth(S, 2, 3); /* density     */

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

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

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

```
      ssSetModelReferenceSampleTimeDefaultInheritance(S);
   }
```

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

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

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

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

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

    /* format the outputs using the reference parameters */
    T[k]   = mxGetScalar(TO(S)) * (double) ftheta;
    P[k]   = mxGetScalar(PO(S)) * (double) fdelta;
    rho[k] = mxGetScalar(RO(S)) * (double) fsigma;
  }
}
```

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

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

13 In the file, read the commented documentation for the following callbacks:

- `mdlInitalizeConditions` — Initializes the state vectors of this S-function.

- `mdlStart` — Initializes the state vectors of this S-function. This function is called once at the start of the model execution.

- `mdlUpdate` — Updates the states of a block.

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

14 Delete or comment out the code for these callbacks.

15 Save the file under another name. For example, save this file as sfun_atmos.c. Do not overwrite the template file.

16 Copy the file sfun_atmos.c into your Fortran working folder, for example, xpc_fortran_test.

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

Compiling and Linking the Wrapper S-Function

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

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

- `libifcoremt.lib`
- `svml_disp.lib`

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

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

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

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

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

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

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

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

Your next task is to validate the Fortran code and wrapper S-function. See "Validating the Fortran Code and Wrapper S-Function" on page 3-13.

Validating the Fortran Code and Wrapper S-Function

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

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

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

1 In the MATLAB window, type

```
fortran_atmos_xpc
```

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

2 Select **Simulation > Run** to simulate the model.

3 Examine the behavior of the Atmosphere model by looking at the signals traced by the Scope block.

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

Preparing the Model for the xPC Target Application Build

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

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

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

This topic assumes that you have validated the Fortran code and wrapper S-function (see "Validating the Fortran Code and Wrapper S-Function" on page 3-13).

1 In the MATLAB window, type

```
fortran_atmos_xpc
```

This opens the Simulink model associated with the Atmosphere model.

2 In the Simulink model, click **Simulation > Model Configuration Parameters**.

The Configuration Parameters dialog box appears.

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

The Code Generation pane opens.

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

5 Click xpctarget.tlc.

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

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

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

7 Click **Apply**.

8 Click **OK**.

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

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

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

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

Building and Running the xPC Target Application

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

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

After the build procedure succeeds, xPC Target automatically downloads the application to the target computer. The Atmosphere model already contains an xPC Target Scope block. This allows you to verify the behavior of the model. You will be able to compare the signals displayed on the target screen with the signals obtained earlier by the Simulink simulation run (see "Validating the Fortran Code and Wrapper S-Function" on page 3-13).

4

Target Application Environment

xPC Target Options Configuration Parameters

The configuration parameters **xPC Target Options** node appears when you select one of the xPC Target settings for the **System target file** parameter in the **Code Generation** pane of the Configuration Parameters dialog box:

- `xpctarget.tlc`

  Generate system target code for xPC Target.

- `xpctargetert.tlc`

  Generate system target code for an xPC Target using the required Embedded Coder® software.

The **xPC Target Options** node allows you to specify how the software generates the target application. Before you create (build) a target application, you might need to enter and select these options. The default values work well for target application creation.

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

- "Introduction to Code Replacement Libraries"
- Code Replacement Library (CRL) and Embedded Targets

For more information on the **xPC Target Options** node, see "Setting Configuration Parameters".

xPC Target Explorer

xPC Target Explorer is a graphical user interface for the xPC Target product. It runs on your host computer and provides a single point of contact for most interactions.

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

In this section...
"Basic Operations" on page 4-4
"Default Target Computers" on page 4-5
"Saving Environment Properties" on page 4-6

Basic Operations

Through xPC Target Explorer, you can perform basic operations, such as:

- Add and configure target computers for the xPC Target software, up to 64 target computers.

- Create boot CDs, removable drives, and network boot images for particular target computers.

- Connect the target computers for your xPC Target system to the host computers.

- Download a prebuilt target application, or DLM, to a target computer.

- Start and stop the application that has been downloaded to the target.

- Add and remove host, target, or file scopes associated with the downloaded target application.

- Monitor signals.

- Add or remove signals associated with xPC Target scopes.

- Start and stop scopes.

- Adjust parameter values for the signals while the target application is running.

To start xPC Target Explorer, type `xpcexplr` in the MATLAB Command Window.

There are four major panes in xPC Target Explorer:

- **Targets** pane — The top-left **Targets** pane lists the targets in your xPC Target hierarchy. Under each target are nodes representing the properties and (if accessible) the file system of the target.

- **Applications** pane — The bottom left **Applications** pane lists the target applications running on the targets. Under each application are nodes representing the properties, signal and parameter groupings, and (if available) the model hierarchy of the application.

- **Scopes** pane — The top right **Scopes** pane lists the scopes defined on the active target applications, whether predefined or dynamically created.

- Center pane — The center pane displays under separate tabs information associated with nodes selected in one of the other panes.

Default Target Computers

When you first start xPC Target Explorer, it opens a default node, `TargetPC1`. You can configure this node for a target computer, then connect the node to the target computer. If you later build a target application from a Simulink model, the xPC Target software builds and downloads that application the default target computer.

You can add other target computer nodes and designate one of them as the default target computer instead of the first one. To set a target computer node as the default, right-click that node and select **Set As Default** from the context-sensitive menu. The default target computer node is boldface.

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

If you want to use the xPC Target command-line interface to work with the target computer, you must indicate which target computer the command is interacting with. If you do not identify a particular target computer, the xPC Target software uses the default target computer.

The target computer environment object, `xpctarget.targets`, manages collective and individual target computer environments. See "Command-Line Setup" on page 4-9.

When you instantiate the target object constructor `xpctarget.xpc` without arguments (for example, `tg=xpc`), the constructor uses the communication properties of the default target computer to communicate with the target computer. The target computer commands `getxpcenv` and `setxpcenv` get and set environment properties for the default target computer only.

Saving Environment Properties

The xPC Target Explorer environment consists of the property settings you define for the **Targets** pane. You can save your settings for the next session.

1 In the MATLAB Command Window, type `xpcexplr`.

2 Set properties in the **Targets** pane.

   After you change a property, the Save icon and menu item are available.

3 Click the Save icon in the toolbar.

   If you do not explicitly save the environment settings, xPC Target Explorer asks on exit if you want to save them.

Command-Line C Compiler Configuration

To configure the host computer for the C compiler using MATLAB language:

1 Install a supported C compiler on the host computer.

For more about the xPC Target C compiler requirements, see
http://www.mathworks.com/support/compilers/current_release/

2 In the MATLAB Command Window, type:

```
xpcsetCC setup
```

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

```
Select your compiler for xPC Target.

[1] Microsoft Visual C++ Compilers 2008 Professional Edition (SP1) in
    c:\Program Files (x86)\Microsoft Visual Studio 9.0
[2] Microsoft Visual C++ Compilers 2010 Professional in
    C:\Program Files (x86)\Microsoft Visual Studio 10.0

[0] None

Compiler:
```

Note The mex -setup command does not affect the C compiler setting
for the xPC Target software.

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

The function verifies that you have selected the required compiler:

```
Verify your selection:

Compiler: Microsoft Visual C++ Compilers 2010 Professional
Location: C:\Program Files (x86)\Microsoft Visual Studio 10.0
```

```
Are these correct [y]/n?
```

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

Command-Line Setup

Use the following procedures to set up the software and hardware configuration for single- and multiple-target systems.

You must have installed and configured a C compiler and verified the target computer BIOS settings. If not, see:

- "Command-Line C Compiler Configuration" on page 4-7.
- "Target Computer BIOS Settings"

1 "Command-Line Ethernet Communication Setup" on page 4-10

2 "Command-Line RS-232 Communication Setup" on page 4-32

3 "Command-Line Target Computer Settings" on page 4-36

4 "Command-Line Target Boot Methods" on page 4-39

The next task is "Run Confidence Test on Configuration".

Command-Line Ethernet Communication Setup

On the host computer, set the properties that your host and target computers require for network communication with multiple target computers. For serial communication, see "Command-Line RS-232 Communication Setup" on page 4-32.

- "Command-Line PCI Bus Ethernet Setup" on page 4-11
- "Command-Line USB-to-Ethernet Setup" on page 4-16

The next task is "Command-Line Target Computer Settings" on page 4-36.

Command-Line PCI Bus Ethernet Setup

If your target computer has a PCI bus, use an Ethernet card for the PCI bus. The PCI bus has a faster data transfer rate than the other bus types.

Follow these procedures:

1 "PCI Bus Ethernet Hardware" on page 4-12

2 "Command-Line PCI Bus Ethernet Settings" on page 4-13

The next task is "Command-Line Target Computer Settings" on page 4-36.

PCI Bus Ethernet Hardware

To install the PCI bus Ethernet card:

1 Acquire a supported PCI bus Ethernet card.

For the most current network communications requirements, see `http://www.mathworks.com/products/xpctarget/-supported-hardware/xPC_Target_Supported_Ethernet_Chipsets.pdf`.

If you want to start the target computer from the network, the Ethernet adapter must be compatible with the **Preboot eXecution Environment (PXE)** specification.

2 Turn off your target computer.

3 If the target computer already has an unsupported Ethernet card, remove the card.

4 Plug the supported Ethernet card into a free PCI bus slot.

5 Assign a static IP address to the target computer Ethernet card.

Although the target computer Ethernet card must have a static IP address, the host computer network adapter card can have a Dynamic Host Configuration Protocol (DHCP) address and can be accessed from the network. When using the product with TCP/IP, you must configure the DHCP server to reserve static IP addresses to prevent these addresses from being assigned to other systems.

6 Connect your target computer Ethernet card to your LAN using an unshielded twisted-pair (UTP) cable.

You can directly connect your computers using a crossover UTP cable with RJ45 connectors. Both computers must have static IP addresses. If the host computer has a second network adapter card, that card can have a DHCP address.

The next task is "Command-Line PCI Bus Ethernet Settings" on page 4-13.

Command-Line PCI Bus Ethernet Settings

After you install the PCI bus Ethernet card, before you can build and download a target application, you must specify the environment properties for the host and target computers.

Before you start, ask your system administrator for the following information for your target computer:

- IP address
- Subnet mask address
- Port number (optional)
- Gateway (optional)

Use the following procedure for target `TargetPC1`:

1 At the MATLAB prompt, get the list of targets and make target `TargetPC1` the default target:

```
tgs=xpctarget.targets;
tgs.makeDefault('TargetPC1');
```

2 Get the environment object for this target computer:

```
env=tgs.Item('TargetPC1');
```

All other settings will be made to this object.

3 Set the host-target communication type to `'TcpIp'`:

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

4 Set the IP address for your target computer. For example:

```
env.TcpIpTargetAddress = '10.10.10.15';
```

5 Set the subnet mask address of your LAN. For example:

```
env.TcpIpSubNetMask = '255.255.255.0';
```

6 Set the TCP/IP port (optional) to a value higher than `'20000'` and less than `'65536'`. For example:

```
env.TcpIpTargetPort = '22222';
```

This property is set by default to `'22222'`, a value higher than the reserved area (`telnet`, `ftp`, and so on).

7 Set the TCP/IP gateway (optional) to the gateway required to access the target computer. For example:

```
env.TcpIpGateway = '255.255.255.255';
```

This property is set by default to `'255.255.255.255'`, which means that you do not use a gateway to connect to your target computer. If you connect your computers with a crossover cable, leave this property as `'255.255.255.255'`.

If you communicate with the target computer from within your LAN, you might not need to change this setting. If you communicate from a host computer located in a LAN different from your target computer (especially via the Internet), you must define a gateway and enter its IP address in this box.

8 Set the bus type to `'PCI'`.

```
env.TcpIpTargetBusType = 'PCI');
```

9 Set the target driver to one of `'3C90x'`, `'I8254x'`, `'I82559'`, `'NS83815'`, `'R8139'`, `'R8168'`, `'Rhine'`, `'RTLANCE'`, or `'Auto'` (the default).

```
env.TcpIpTargetDriver = 'Auto';
```

For target driver `'Auto'`, the software determines the target computer TCP/IP driver from the card installed on the target computer. If no supported Ethernet card is installed in your target computer, the software returns an error.

10 If the target computer has multiple Ethernet cards, follow the procedure in "Command-Line Ethernet Card Selection by Index" on page 4-29.

11 Save the changes to your environment:

```
tgs.save
```

Repeat this procedure as required for each target computer.

The next task is "Command-Line Target Computer Settings" on page 4-36.

Command-Line USB-to-Ethernet Setup

If the target computer has a USB 2.0 port but no supported PCI or ISA Ethernet card, use a USB-to-Ethernet adapter.

Follow these procedures:

1 "USB-to-Ethernet Hardware" on page 4-17

2 "Command-Line USB-to-Ethernet Settings" on page 4-19

The next task is "Command-Line Target Computer Settings" on page 4-36.

USB-to-Ethernet Hardware

To install the USB-to-Ethernet adapter:

1 Acquire a supported USB-to-Ethernet adapter.

For the most current network communications requirements, see `http://www.mathworks.com/products/xpctarget/-supported-hardware/xPC_Target_Supported_Ethernet_Chipsets.pdf`.

If you want to start the target computer from the network, the Ethernet adapter must be compatible with the **Preboot eXecution Environment (PXE)** specification.

2 Turn off your target computer.

3 Plug an Ethernet-to-USB adapter into the USB port on the target.

4 Connect the Ethernet-to-USB adapter to your LAN using an unshielded twisted-pair (UTP) cable.

5 Assign a static IP address to the target computer USB-to-Ethernet adapter.

Although the target computer Ethernet adapter must have a static IP address, the host computer network adapter can have a Dynamic Host Configuration Protocol (DHCP) address and can be accessed from the network. When using the product with TCP/IP, you must configure the DHCP server to reserve static IP addresses to prevent these addresses from being assigned to other systems.

6 Connect your target computer Ethernet adapter to your LAN using an unshielded twisted-pair (UTP) cable.

You can directly connect your computers using a crossover UTP cable with RJ45 connectors. Both computers must have static IP addresses. If the host computer has a second network adapter, that adapter can have a DHCP address.

Do not connect the host computer USB port to the target computer USB port using a USB cable. A USB-to-Ethernet adapter plugged into the

target computer USB port behaves like an Ethernet card installed on the target computer.

The next task is "Command-Line USB-to-Ethernet Settings" on page 4-19.

Command-Line USB-to-Ethernet Settings

After you install the USB-to-Ethernet adapter, before you can build and download a target application, you must specify the environment properties for the host and target computers.

Before you start, ask your system administrator for the following information for your target computer:

- IP address
- Subnet mask address
- Port number (optional)
- Gateway (optional)

Use the following procedure for target `TargetPC1`:

1 At the MATLAB prompt, get the list of targets and make target `TargetPC1` the default target:

```
tgs=xpctarget.targets;
tgs.makeDefault('TargetPC1');
```

2 Get the environment object for this target computer:

```
env=tgs.Item('TargetPC1');
```

All other settings will be made to this object.

3 Set the host-target communication type to `'TcpIp'`:

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

4 Set the IP address for your target computer. For example:

```
env.TcpIpTargetAddress = '10.10.10.15';
```

5 Set the subnet mask address of your LAN. For example:

```
env.TcpIpSubNetMask = '255.255.255.0';
```

6 Set the TCP/IP port (optional) to a value higher than `'20000'` and less than `'65536'`. For example:

```
env.TcpIpTargetPort = '22222';
```

This property is set by default to `'22222'`, a value higher than the reserved area (`telnet`, `ftp`, and so on).

7 Set the TCP/IP gateway (optional) to the gateway required to access the target computer. For example:

```
env.TcpIpGateway = '255.255.255.255';
```

This property is set by default to `'255.255.255.255'`, which means that you do not use a gateway to connect to your target computer. If you connect your computers with a crossover cable, leave this property as `'255.255.255.255'`.

If you communicate with the target computer from within your LAN, you might not need to change this setting. If you communicate from a host computer located in a LAN different from your target computer (especially via the Internet), you must define a gateway and enter its IP address in this box.

8 Set the bus type to `'USB'`.

```
env.TcpIpTargetBusType = 'USB';
```

9 Set the target driver to one of `'USBAX772'`, `'USBAX172'`, or `'Auto'`.

```
env.TcpIpTargetDriver = 'Auto';
```

If the target driver is `'Auto'`, the software sets the driver to `'USBAX772'`, the driver most commonly used.

10 Save the changes to your environment:

```
tgs.save
```

Repeat this procedure as required for each target computer.

The next task is "Command-Line Target Computer Settings" on page 4-36.

Command-Line ISA Bus Ethernet Setup

Your target computer might not have an available PCI bus slot or USB 2.0 port. In these cases, use an Ethernet card for an ISA bus.

Note Host-target communication using ISA bus Ethernet adapters will be removed in a future release. Use PCI or USB bus adapters instead.

---

1 "ISA Bus Ethernet Hardware" on page 4-22

2 "Command-Line ISA Bus Ethernet Settings" on page 4-24

The next task is "Command-Line Target Computer Settings" on page 4-36.

ISA Bus Ethernet Hardware

To install an ISA bus Ethernet card, perform the following steps:

Note Host-target communication using ISA bus Ethernet adapters will be removed in a future release. Use PCI or USB bus adapters instead.

---

1 Acquire a supported ISA bus Ethernet card.

For the most current network communications requirements, see `http://www.mathworks.com/products/xpctarget/-supported-hardware/xPC_Target_Supported_Ethernet_Chipsets.pdf`.

If you want to start the target computer from the network, the Ethernet adapter must be compatible with the **Preboot eXecution Environment (PXE)** specification.

2 Turn off your target computer.

3 On your ISA bus card, assign an IRQ and I/O-port base address by moving the jumpers or switches on the card. Write down these settings, because you must enter them in xPC Target Explorer.

Set the IRQ line to `5` and the I/O-port base address to around `0x300`. If one of these hardware settings leads to a conflict in your target computer, select another IRQ or I/O-port base address.

If your ISA bus card does not contain jumpers to set the IRQ line and the base address, after installation use the utility on the installation disk supplied with your card to manually assign the IRQ line and base address.

If you use an Ethernet card for an ISA bus within a target computer that has a PCI bus, after installation you must reserve the chosen IRQ line number for the Ethernet card in the PCI BIOS. To set up the PCI BIOS, refer to your BIOS setup documentation.

Do not configure the card as a PnP-ISA device.

4 If the target computer already has an unsupported Ethernet card, remove the card. Plug the compatible network card into a free ISA bus slot.

5 Assign a static IP address to the target computer Ethernet card.

Although the target computer Ethernet card must have a static IP address, the host computer network adapter card can have a Dynamic Host Configuration Protocol (DHCP) address and can be accessed from the network. When using the product with TCP/IP, you must configure the DHCP server to reserve static IP addresses to prevent these addresses from being assigned to other systems.

6 Connect your target computer Ethernet card to your LAN using an unshielded twisted-pair (UTP) cable.

You can directly connect your computers using a crossover UTP cable with RJ45 connectors. Both computers must have static IP addresses. If the host computer has a second network adapter card, that card can have a DHCP address.

The next task is "Command-Line ISA Bus Ethernet Settings" on page 4-24.

Command-Line ISA Bus Ethernet Settings

After you install the ISA bus Ethernet card, before you can build and download a target application, you must specify the environment properties for the host and target computers.

Before you start, ask your system administrator for the following information for your target computer:

- IP address
- Subnet mask address
- Port number (optional)
- Gateway (optional)

Use the following procedure for target TargetPC1:

1 At the MATLAB prompt, get the list of targets and make target TargetPC1 the default target:

```
tgs=xpctarget.targets;
tgs.makeDefault('TargetPC1');
```

2 Get the environment object for this target computer:

```
env=tgs.Item('TargetPC1');
```

All other settings will be made to this object.

3 Set the host-target communication type to 'TcpIp':

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

4 Set the IP address for your target computer. For example:

```
env.TcpIpTargetAddress = '10.10.10.15';
```

5 Set the subnet mask address of your LAN. For example:

```
env.TcpIpSubNetMask = '255.255.255.0';
```

6 Set the TCP/IP port (optional) to a value higher than `'20000'` and less than `'65536'`. For example:

```
env.TcpIpTargetPort = '22222';
```

This property is set by default to `'22222'`, a value higher than the reserved area (`telnet`, `ftp`, and so on).

7 Set the TCP/IP gateway (optional) to the gateway required to access the target computer. For example:

```
env.TcpIpGateway = '255.255.255.255';
```

This property is set by default to `'255.255.255.255'`, which means that you do not use a gateway to connect to your target computer. If you connect your computers with a crossover cable, leave this property as `'255.255.255.255'`.

If you communicate with the target computer from within your LAN, you might not need to change this setting. If you communicate from a host computer located in a LAN different from your target computer (especially via the Internet), you must define a gateway and enter its IP address in this box.

8 Set the bus type to `'ISA'`.

```
env.TcpIpTargetBusType = 'ISA';
```

9 Set the target driver to one of `'NE2000'` or `'SMC91C9X'`. For example:

```
env.TcpIpTargetDriver = 'NE2000';
```

Target driver `'Auto'` is not supported for bus type `'ISA'`.

10 Set the I/O-port base address and IRQ to values that correspond with the jumper settings or ROM settings on your ISA bus Ethernet card. For example:

```
env.TcpIpTargetISAMemPort = '0x300';
env.TcpIpTargetISAIRQ = '5';
```

11 Save the changes to your environment:

4-25

```
tgs.save;
```

Repeat this procedure as required for each target computer.

The next task is "Command-Line Target Computer Settings" on page 4-36.

Ethernet Card Selection by Index

If the target computer has multiple Ethernet cards, you must specify which card to use for host-target communication. Use the following procedure to discover the Ethernet index of the PCI cards on the target computer and to specify which card to use.

Note For this procedure, you must be able to burn CDs on your host computer and use `Network` boot mode for routine target operations.

Use the following procedure for target `TargetPC1`:

1 In the MATLAB window, type:

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

2 At the MATLAB prompt, type:

```
env.ShowHardware = 'on';
```

3 At the MATLAB prompt, type: `xpcexplr`.

4 In the **Targets** pane, expand the target computer node.

5 In the toolbar, click the Target Properties icon .

6 Select **Host-to-Target communication** and set **Target driver** to `Auto`. If you set **Target driver** to a specific driver, such as `INTEL_I82559`, the kernel displays only information about boards that use that driver.

7 Select **Target settings** and clear the **Graphics mode** check box. This setting causes the kernel to print text only.

8 Select **Boot configuration** and set **Boot mode** to `CD`.

9 Click **Create boot disk** and follow the prompts to create a new boot disk.

10 Insert the new boot disk and restart the target computer from the computer boot switch.

After the start is complete, the target monitor displays information about the Ethernet cards in the target computer, for example :

```
index: 0, driver: RTLANCE, Bus: 16, Slot: 7, Func: 0
index: 1, driver: R8139, Bus: 16, Slot: 8, Func: 0
index: 2, driver: I82559, Bus: 16, Slot: 9, Func: 0
```

You might need to change the boot order from the target computer BIOS to allow starting from your disk. After the kernel starts with `ShowHardware` `'on'`, the host computer cannot communicate with the target computer.

11 Note the index of the Ethernet card that you want to use for host-target communication, for example, `2`.

12 At the MATLAB prompt, type:

```
env.ShowHardware = 'off';
env.EthernetIndex = '#';
```

`#` is the index of the Ethernet card, for example, `2`.

13 Select **Target settings** and select the **Graphics mode** check box.

14 Set **Boot mode** to `Network`.

15 Click **Create boot disk**.

16 Remove the boot disk from the target computer drive and start the target computer from the computer boot switch.

The kernel selects the specified Ethernet card as the target computer card, instead of the default card with index number `0`.

Repeat this procedure as required for each target computer.

Command-Line Ethernet Card Selection by Index

If you are using multiple target computers that have multiple Ethernet cards, you must specify which card to use for host-target communication. Use the following procedure to discover the Ethernet index of the PCI cards on a specific target and specify which card to use.

Note For this procedure, you must be able to burn CDs on your host computer and use network boot mode for routine target operations.

Use the following procedure for target `TargetPC1`:

1 In the MATLAB window, type

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

2 At the MATLAB prompt, type:

```
env.ShowHardware = 'on';
```

3 Set the Ethernet driver to the default:

```
env.TcpIpTargetDriver = 'Auto';
```

If `TcpIpTargetDriver` is set to a specific driver, such as `'I82559'`, the kernel displays only information about boards that use that driver.

4 Set the boot method to CD/DVD boot:

```
env.TargetBoot='CDBoot';
```

5 Set the target monitor to print text only:

```
env.TargetScope = 'Disabled' ;
```

6 Save the changes to your environment:

```
tgs.save
```

7 Type `xpcbootdisk`.

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

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

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

8 Insert the new boot disk and restart the target computer from the computer boot switch.

After the start is complete, the target monitor displays information about the Ethernet cards in the target computer, for example:

```
index: 0, driver: RTLANCE, Bus: 16, Slot: 7, Func: 0
index: 1, driver: R8139, Bus: 16, Slot: 8, Func: 0
index: 2, driver: I82559, Bus: 16, Slot: 9, Func: 0
```

You might need to change the boot order from the target computer BIOS to allow starting from your disk. After the kernel starts with `ShowHardware` `'on'`, the host computer cannot communicate with the target computer.

9 Note the index of the Ethernet card you want to use for host-target communication, for example, 2.

10 At the MATLAB prompt, type:

```
env.ShowHardware = 'off';
env.EthernetIndex = '#';
```

is the index of the Ethernet card, for example, 2.

11 Set the boot method back to network boot:

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

12 Set the target monitor to graphics mode:

```
env.TargetScope = 'Enabled' ;
```

13 Save the changes to your environment:

```
tgs.save
```

14 Type `xpcnetboot`.

15 Start the target computer from the computer boot switch.

The kernel selects the specified Ethernet card as the target computer card, instead of the default card with index number `0`.

Repeat this procedure as required for each target computer.

Command-Line RS-232 Communication Setup

On the host computer, set the properties that your host and target computers require for serial communication with a single target computer. For network communication, see "Command-Line Ethernet Communication Setup" on page 4-10.

Note RS-232 Host-Target communication mode will be removed in a future release. Use Ethernet instead.

- "RS-232 Hardware" on page 4-33
- "Command-Line RS-232 Settings" on page 4-34

The next task is "Command-Line Target Computer Settings" on page 4-36.

RS-232 Hardware

Before you can use serial communication for host-target communication, you must install the following RS-232 hardware:

1 Acquire a null modem cable:
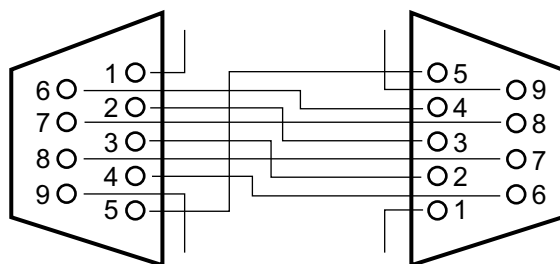
DB9 Female   DB9 Female

2 Connect the host and target computers with the null modem cable, using either the COM1 or COM2 port.

Note which port is in use on the host computer. You need to set the host computer port in the environment property settings.

The next task is "Command-Line RS-232 Settings" on page 4-34.

4-33

Command-Line RS-232 Settings

After you have installed the serial communication hardware, before you can build and download a target application, specify the environment properties for the host and target computers.

- Do not use host scopes and a scope viewer on the host computer to acquire and display large blocks of data. The slowness of the RS-232 connection causes large delays for large blocks of data.

- When you use serial communication, boot mode type `'NetworkBoot'` is not supported.

Use the following procedure for target `TargetPC1`:

1 At the MATLAB prompt, get the list of targets and make target `TargetPC1` the default target:

```
tgs=xpctarget.targets;
tgs.makeDefault('TargetPC1');
```

2 Get the environment object for this target computer:

```
env=tgs.Item('TargetPC1');
```

All other settings will be made to this object.

3 At the MATLAB prompt, set the host-target communication type to `'RS232'`:

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

4 For host port, select one of `'COM1'` or `'COM2'`. For example:

```
env.RS232HostPort = 'COM1';
```

The default is `'COM1'`. xPC Target selects the target computer port automatically.

5 Select a baud rate as high as possible. For example:

```
env.RS232Baudrate = '115200';
```

The default is `115200`. A baud rate less than 38400 can cause communication failures.

6 Save the changes to your environment:

```
tgs.save;
```

Repeat this procedure as required for each target computer.

The next task is "Command-Line Target Computer Settings" on page 4-36.

Command-Line Target Computer Settings

To run an xPC Target model on a target machine, you must configure the target settings to match the capabilities of the target machine.

Use the following procedure for target `TargetPC1`:

1 At the MATLAB prompt, get the list of targets and make target `TargetPC1` the default target:

```
tgs=xpctarget.targets;
tgs.makeDefault('TargetPC1');
```

2 Get the environment object for this target computer. All other settings will be made to this object.

```
env=tgs.Item('TargetPC1');
```

3 Assign the following target computer settings as required:

- **Target scope display.**
  - `env.TargetScope = 'Enabled'` (the default) — Use if you want to display information, such as a target scope, in graphic format.
  - `env.TargetScope = 'Disabled'` — Use if you want to display information as text.

  To use the full features of a target scope, install a keyboard on the target computer.

- **USB support.**
  - `env.USBSupport = 'on'` (the default) — Use if you want to use a USB port on the target computer; for example, to connect a USB mouse.
  - `env.USBSupport = 'off'` — Otherwise.

- **Secondary IDE support.**
  - `env.SecondaryIDE = 'on'` — Use only if you want to use the disks connected to a secondary IDE controller.
  - `env.SecondaryIDE = 'off'` (the default) — Otherwise.

- **Multicore support.**

env.MulticoreSupport = 'on' — Use if your target computer has multicore processors that you want to take advantage of.

env.MulticoreSupport = 'off' (the default) — Otherwise.

- **Non-Pentium support.**

  env.NonPentiumSupport = 'on' — Use if your target computer has a 386 or 486 compatible processor.

  env.NonPentiumSupport = 'off' (the default) — Otherwise.

  If your target computer has a Pentium or higher compatible processor, setting this value to 'on' slows the performance of your target computer.

- **Target RAM size.**

  env.TargetRAMSizeMB = 'Auto' (the default) — Use if you want the target application to read the target computer BIOS and determine the amount of memory up to a maximum of 2 GB.

  env.TargetRAMSizeMB = 'xxx' — Use if the target application cannot read the BIOS. You must assign the amount of memory, in megabytes, up to a maximum of 2 GB.

  The Target RAM size parameter defines the total amount of installed RAM in the target computer available for the kernel, target application, data logging, and other functions that use the heap.

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

- **Maximum model size.**

  env.MaxModelSize = '1MB' (the default) — Use in Stand Alone mode if the target application requires at most this much memory on the target computer.

  env.MaxModelSize = '4MB' — Otherwise.

  Memory not used by the target application is used by the kernel and by the heap for data logging. Selecting too high a value leaves less memory for data logging. Selecting too low a value does not reserve enough memory for the target application and creates an error. You can approximate the size of the target application by the size of the DLM file produced by the build process.

4 Save the changes to your environment:

```
tgs.save;
```

Repeat this procedure as required for each target computer.

The next task is "Command-Line Target Boot Methods" on page 4-39.

Command-Line Target Boot Methods

You can start your target computer with the xPC Target kernel using one of several methods.

xPC Target Turnkey systems come with DOS Loader software preinstalled. You can set up the DOS Loader boot method on your host or configure another boot method. For more information, see your xPC Target Turnkey system user documentation.

1 Before creating a boot kernel, perform "Command-Line Kernel Creation Prechecks" on page 4-40.

2 Select one of the following methods:

- "Command-Line Network Boot Method" on page 4-41
- "Command-Line CD/DVD Boot Method" on page 4-43
- "Command-Line DOS Loader Boot Method" on page 4-45
- "Command-Line Removable Disk Boot Method" on page 4-47
- "Command-Line Stand Alone Boot Method" on page 4-49.

3 If you select Stand Alone mode, restart the target computer and test your application in Stand Alone mode. The confidence test is not intended for standalone execution. You must create and execute your own confidence test for standalone mode.

Otherwise, the next task is "Run Confidence Test on Configuration".

Command-Line Kernel Creation Prechecks

Before creating the target boot kernel, configure your xPC Target system. At a minimum, do the following:

1 Check the physical connections between the host computer and the target computer. If you are using TCP/IP, these are Ethernet connections that may pass through a LAN.

2 Check your target computer BIOS settings (see "Target Computer BIOS Settings").

3 Check that you have write permission for your current working folder.

4 In the MATLAB command window, type:

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

The contents of environment object `env` are printed in the command window.

5 Check the host-to-target communication settings. As required, see:

- "Command-Line Ethernet Communication Setup" on page 4-10
- "Command-Line RS-232 Communication Setup" on page 4-32

6 Check that `TargetBoot` is set to the required value.

Repeat this procedure as required for each target computer.

Command-Line Network Boot Method

After you have configured the target computer environment parameters, you can use a dedicated Ethernet network to load and run the xPC Target kernel. You do not need a boot CD or removable boot drive.

There are the following limitations:

- Do not use the network boot method on a corporate or nondedicated network. Doing so might interfere with dynamic host configuration protocol (DHCP) servers and cause problems with the network.

- Your Ethernet card must be compatible with the Preboot eXecution Environment (PXE) specification.

- If the target computer and host computer communicate by serial communication (RS-232), you cannot start the target computer across the network.

- If Stand Alone mode is enabled, you cannot start the target computer across the network.

Before you start, establish the required Ethernet connection between host and target using the procedure in "Command-Line Ethernet Communication Setup" on page 4-10.

Use the following procedure for target `TargetPC1`:

1 In the MATLAB command window, type:

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

The contents of environment object `env` are printed in the command window. Some properties may already have the required values.

2 Set network boot method:

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

3 Set a TCP/IP address. Verify that the subnet of this IP address is the same as the host computer. Otherwise your network boot fails. For example, type:

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

4 Set the target computer MAC address (in hexadecimal). For example, type:

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

5 Save the changes to your environment:

```
tgs=xpctarget.targets;
tgs.save
```

6 In the MATLAB Command Window, type:

```
xpcnetboot
```

The following message appears:

```
Current boot mode: NetworkBoot
```

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

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

Repeat this procedure as required for each target computer.

The next task is "Run Confidence Test on Configuration".

Command-Line CD/DVD Boot Method

After you have configured the target computer environment parameters, you can use a target boot CD or DVD to load and run the xPC Target kernel. This topic describes using the MATLAB command line to create a boot CD or DVD for a single target computer system. To use this capability, your host computer must run under one of the following Windows systems:

- Microsoft Windows 7

- Microsoft Windows Vista™

- Microsoft Windows XP Service Pack 2 or 3 with Image Mastering API v2.0 (IMAPIv2.0), available at `http://support.microsoft.com/kb/KB932716`.

Use the following procedure for target `TargetPC1`:

1 In the MATLAB window, type:

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

2 Set the CD boot method:

```
env.TargetBoot='CDBoot'
```

3 Save the changes to your environment:

```
tgs=xpctarget.targets;
tgs.save
```

4 In the MATLAB window, type `xpcbootdisk`.

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

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

Insert an empty CD/DVD. Available drives:
  [1]  d:\
```

```
[0]  Cancel Burn
```

5 Insert the empty CD or DVD in the host computer.

6 Type 1 and then press **Enter**.

7 When the write operation has finished, remove the CD or DVD from the drive.

8 Insert the bootable CD/DVD into your target computer drive and restart the target computer.

Repeat this procedure as required for each target computer.

The next task is "Run Confidence Test on Configuration".

Command-Line DOS Loader Boot Method

DOSLoader mode allows you to start the xPC Target kernel on a target computer from a fixed or removable device with DOS boot capability, such as a hard disk or flash memory. After starting the target computer, you can download your application from the host computer over a serial or network connection between the host and target computers.

Note To run in DOSLoader mode, the target computer boot device must provide a minimal DOS environment complying with certain restrictions. For details, see:

- "Create a DOS System Disk"
- "DOS Loader Mode Restrictions"

Use the following procedure for target `TargetPC1`:

1 For a specific target computer, retrieve the specific target computer environment object:

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

2 Set the DOS Loader boot method:

```
env.TargetBoot = 'DOSLoader';
```

3 Set `DOSLoaderLocation` to the folder where you want to create the DOSLoader boot files. This location can be a local folder on the host computer or a removable storage device that you use to start the target computer. By default, the folder is the current working folder.

```
env.DOSLoaderLocation = 'D:\';
```

4 Save the changes to your environment:

```
tgs=xpctarget.targets;
```

```
tgs.save
```

5 In the MATLAB Command Window, type `xpcbootdisk`.

The xPC Target software displays the following message:

```
Current boot mode: DOSLoader
xPC Target DOS Loader files are successfully created
```

This operation creates the following boot files in the specified location:

```
autoexec.bat
xpcboot.com
*.rtb
```

6 If you create boot files on a local hard disk, copy these files to a floppy disk, CD/DVD, or other removable storage media.

7 Transfer the boot files to your target computer or insert the removable media containing the boot files into the target computer drive or USB port.

8 Verify that `autoexec.bat` file is on the DOS boot path (typically the root folder).

9 Select the required boot device in the BIOS of the target computer.

10 Start the target computer.

When the target computer starts, it loads DOS, which executes the `autoexec.bat` file. This file starts the xPC Target kernel (*.rtb). The target computer then awaits commands from the host computer.

Repeat this procedure as required for each target computer.

The next task is "Run Confidence Test on Configuration".

Command-Line Removable Disk Boot Method

After you have configured the target computer environment parameters, you can use a target boot floppy disk, removable drive, or USB flash drive to load and run the xPC Target kernel. This topic describes using the MATLAB command line to create a removable boot disk.

If you are creating a removable boot drive from a USB flash drive, you must create a bootable partition on the drive before performing this procedure. See "Create a Bootable Partition".

Use the following procedure for target `TargetPC1`:

1 In the MATLAB command window, type:

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

2 In the output of the Item command, verify that property `TargetBoot` is `BootFloppy`.

If required, update property `TargetBoot`, for instance by using the command `env.TargetBoot='BootFloppy'`.

3 Save the changes to your environment:

```
tgs=xpctarget.targets;
tgs.save
```

4 If you are creating a removable boot disk from a USB drive, insert the USB drive in the host computer USB port and wait for it to be recognized.

5 In the MATLAB command window, type `xpcbootdisk`.

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

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

6 If required, insert an empty removable disk in the host computer drive and then press a key.

7 When the write operation has finished, remove the removable disk from the drive or USB port.

8 Insert the removable boot disk into your target computer drive or USB port and restart the target computer.

Repeat this procedure as required for each target computer.

The next task is "Run Confidence Test on Configuration".

Command-Line Stand Alone Boot Method

The xPC Target Embedded Option™ software extends the xPC Target base product with Stand Alone mode. For more information on Stand Alone mode, see "Stand Alone Mode Embedded Option".

The target computer and its DOS environment must meet specific requirements to run in Stand Alone mode:

• "Stand Alone Target Computer Prechecks"

• "Stand Alone Mode Restrictions"

To set up your target computer for Stand Alone mode:

1 "Command-Line Stand Alone Settings" on page 4-50

2 "Stand Alone Target Application Build"

3 "Stand Alone Target Application Transfer"

4 "Stand Alone Target Application Boot Configuration"

Continue by restarting the target computer and testing your application in Stand Alone mode.

Command-Line Stand Alone Settings

Use the command line to set the kernel environment properties. When you are done, you can create a Stand Alone kernel/target application.

For Stand Alone mode, you do not create an xPC Target boot disk or network boot image. Instead, you copy files created from the build process to the target computer hard drive.

Use the following procedure for target `TargetPC1`:

1 In the MATLAB command window, type:

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

The contents of environment object `env` are printed in the command window.

2 Set network boot method:

```
env.TargetBoot='StandAlone';
```

3 Save the changes to your environment:

```
tgs.save
```

Repeat this procedure as required for each target computer.

The next task is "Stand Alone Target Application Build".

5

Signals and Parameters

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

Signal Monitoring Basics

Signal monitoring acquires real-time signal data without time information during target application execution. There is minimal additional load on the real-time tasks. Use signal monitoring to acquire signal data without creating scopes that run on the target computer.

In addition to signal monitoring, xPC Target enables you to monitor Stateflow states as test points through the xPC Target Explorer and MATLAB command-line interfaces. You designate data or a state in a Stateflow diagram as a test point, making it observable during execution. You can work with Stateflow states as you do with xPC Target signals, such as monitoring or plotting Stateflow states.

When you monitor signals from referenced models, you must first set the test point for the signal in the referenced model. The software ignores signal labels in referenced models.

Note

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

- Some signals are not observable. See "Nonobservable Signals and Parameters" on page 5-140.

You can monitor signals using xPC Target Explorer and MATLAB language. You can monitor Stateflow states using xPC Target Explorer, MATLAB language, and Simulink External Mode.

Monitor Signals Using xPC Target Explorer

This procedure uses the model `xpcosc` as an example. You must have already completed the following setup:

1 Built and downloaded the target application to the target computer using Simulink (⁘ on the toolbar).

2 Run xPC Target Explorer (command `xpcexplr`).

3 Connected to the target computer in the **Targets** pane (🔵 on the toolbar).

4 Set property **Stop time** to `inf` in the **Applications** pane (⚙ on the toolbar).

To monitor a signal:

1 In xPC Target Explorer, expand the **Model Hierarchy** node under the target application node.

2 To view the signals in the target application, select the model node. On the toolbar, click the View Signals icon ⴹ. The Signals workspace opens.

3 To view the value of a signal, in the Signals workspace, select the **Monitor** check box for the signal. For instance, select the check boxes for **Signal Generator** and **Integrator1**. The signal values are shown in the **Monitoring Value** column.

4 To start execution, click the target application. On the toolbar, click the Start icon ▶.

5 To stop execution, click the target application. On the toolbar, click the Stop icon ⬛.

The Application Parameters and Signals workspaces look like this figure.

- To group signals, see "Create Signal Groups Using xPC Target Explorer" on page 5-54.

- When you are monitoring a signal group, you can change the output format of the group by selecting one of the options in the **Format** column.

- To make both workspaces visible at the same time, click and hold the tab for one workspace and drag it down until the following icon appears in the middle of the dialog box: . Continue to drag until the cursor reaches the required quadrant, and then release the mouse button.

- If a block name consists only of spaces, xPC Target Explorer does not display a node for or signals from that block. To reference such a block, provide an alphanumeric name for that block, rebuild and download the model to the target computer, and reconnect the MATLAB session to the target computer.

Monitor Signals Using MATLAB Language

This procedure uses the model xpc_osc3 as an example. You must have already completed the following setup:

1 Built and downloaded the target application to the target computer using Simulink (on the toolbar).

2 Assigned tg to the target computer.

1 To get a list of signals, type:

```
tg.ShowSignals='on'

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

If your signal has a unique label, its label is displayed in the Label column. If the label is not unique, the command returns an error. If the signal label is in a referenced model, the software ignores it.

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

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

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

```
ans=
  3.731
```

See also "Configure Target Scopes Using MATLAB Language" on page 5-67.

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

Configure Stateflow States as Test Points

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

1 In the MATLAB window, type old_sf_car.

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

3 In the Configuration Parameters dialog box, click the **Code Generation** node.

4 To build a basic target application, in the **Target selection** section of the **Code Generation** pane, click **Browse** at the **System target file** list. Click xpctarget.tlc, and then click **OK**.

5 In the old_sf_car model, double-click the shift_logic chart.
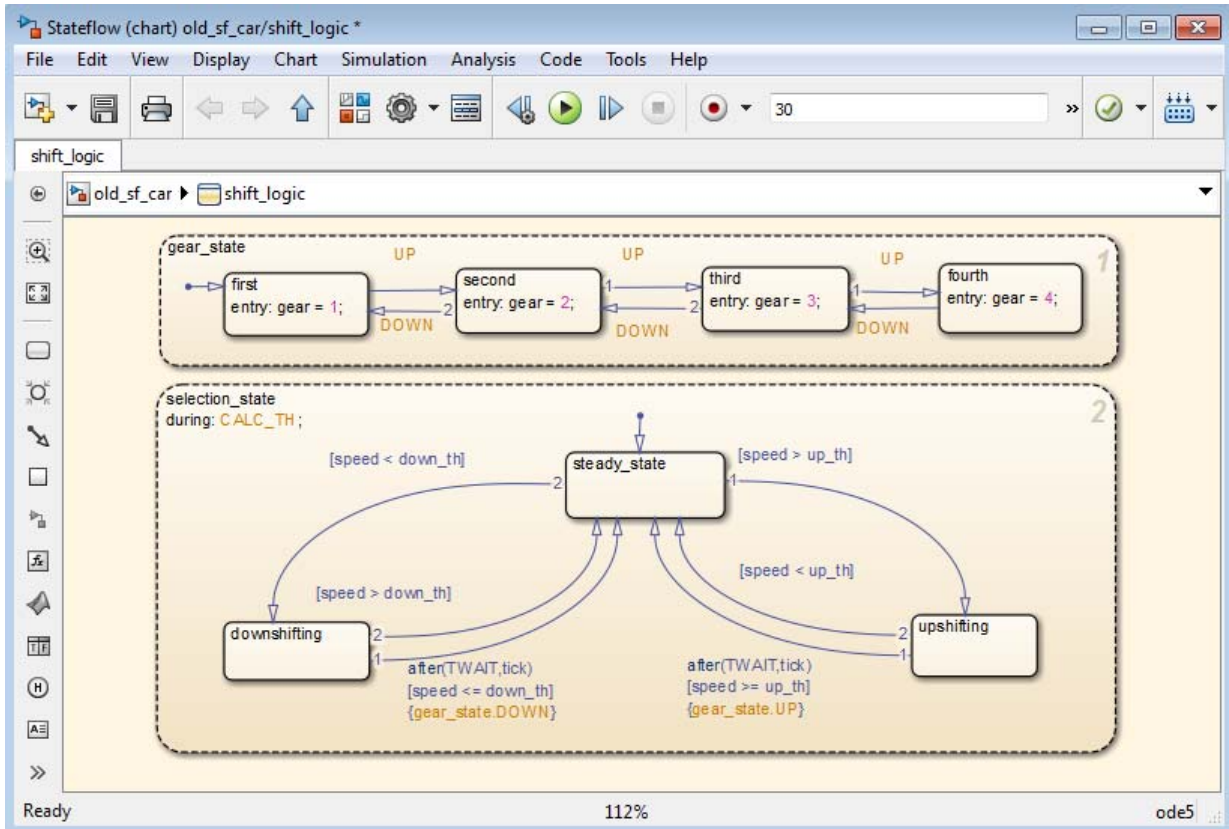
6 In the shift_logic chart, click **Tools > Model Explorer**.

7 In the Model Explorer, expand **old_sf_car**, and then **shift_logic**.

8 Expand **gear_state**, and then select **first**.

9 In the **State first** pane **Logging** tab, select the **Test point** check box. This selection creates a test point for the first state.

10 Click **Apply**.

11 Build and download the `old_sf_car` target application to the target computer.

12 View Stateflow states using one of:

- "Monitor Stateflow States Using xPC Target Explorer" on page 5-12
- "Monitor Stateflow States Using MATLAB Language" on page 5-15
- "Animate Stateflow Charts Using Simulink External Mode" on page 5-16

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

Monitor Stateflow States Using xPC Target Explorer

You must have already and built and downloaded the application to carry out this procedure.

This procedure uses the model old_sf_car as an example. You must have already completed the following setup:

1 Set Stateflow states as test points.

2 Built and downloaded the target application to the target computer using Simulink (on the toolbar).

3 Run xPC Target Explorer (command xpcexplr).

4 Connected to the target computer in the **Targets** pane (on the toolbar).

To monitor a test point:

1 In the **Applications** pane, expand the target application and the **Model Hierarchy** node.

2 To view the test point, select shift_logic and click the View Signals icon on the toolbar.

The Signals workspace opens. The test point gear_state.first appears like other signals in the Signals workspace.

3 In the Signals workspace, select the **Monitor** check box for gear_state.first. The value of the signal is shown in the **Monitoring Value** column.

The Signals workspace looks like this figure.
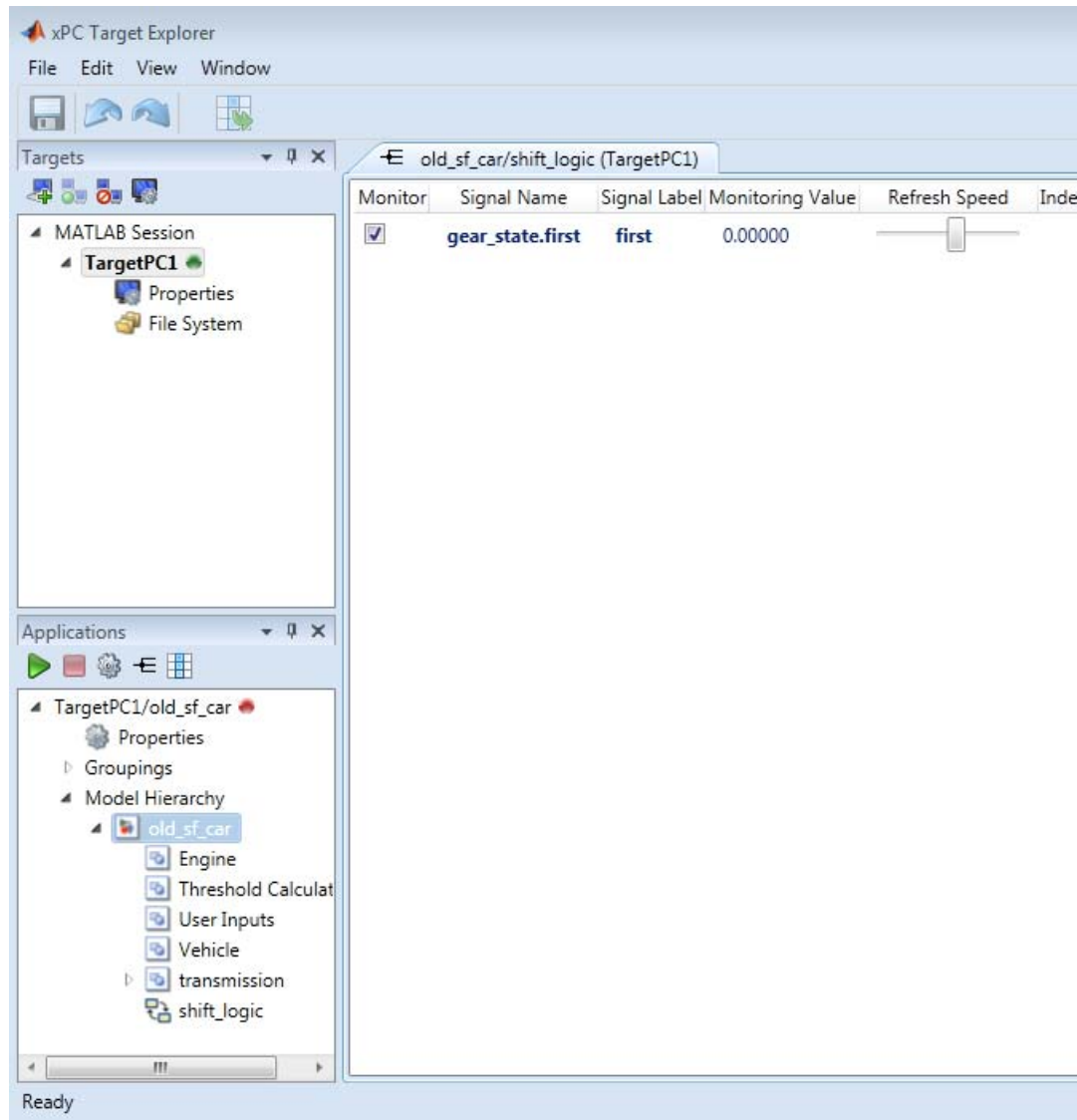
4 To start execution, click the target application. On the toolbar, click the Start icon ▶.

5 To stop execution, click the target application. On the toolbar, click the Stop icon ⬛.

To group signals, see "Create Signal Groups Using xPC Target Explorer" on page 5-54.

Tip When you are monitoring a signal group, you can change the output format of the group by selecting one of the options in the **Format** column.

Monitor Stateflow States Using MATLAB Language

You must have already set Stateflow states as test points. If you have not, see "Configure Stateflow States as Test Points" on page 5-9.

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

```
tg=xpc
```

2 To display the signals in the target application, type:

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

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

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

```
shift_logic:gear_state.first
```

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

Animate Stateflow Charts Using Simulink External Mode

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

You must be familiar with the use of Stateflow animation. For more information on Stateflow animation, see "Animate Stateflow Charts" in the Stateflow documentation.

1 In the Simulink Editor window, select **Simulation > Mode > External**.

2 Select **Code > External Mode Control Panel**.

3 Select **Signal & Triggering**.

4 In the Trigger section of the External Signal & Triggering window:

- Set **Mode** to normal.
- In the **Duration** box, enter 5.
- Select the **Arm when connecting to target** check box.

5 Click **Apply**.

6 Select **Simulation > Model Configuration Parameters**.

7 Navigate to the **xPC Target options** node.

8 Select the **Enable Stateflow animation** check box.

9 Click **Apply**.

10 Build and download the model to the target computer.

11 On the toolbar, click the Connect To Target icon ⊚.

12 To start the simulation, click the Run icon ▶ on the toolbar.

The simulation begins to run. You can observe the animation by opening the Stateflow Editor for your model.

13 To stop the simulation, click the Stop icon on the toolbar.

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

Signal Tracing Basics

Signal tracing acquires signal and time data from a target application. You can then visualize the data on the target computer or upload the data and visualize it on the host computer while the target application is running.

You trace signals using target and host scopes and view them using xPC Target Explorer, Simulink External Mode, MATLAB language, and a Web browser interface.

xPC Target Explorer can display multidimensional signals in column-major format.

Some signals are not observable. See "Nonobservable Signals and Parameters" on page 5-140.

Configure Target Scope (xPC) Blocks

xPC Target includes a specialized Scope (xPC) block that you can configure to display signal and time data on the target computer monitor. To do this, add a Scope (xPC) block to the model, select **Scope type** Target, and configure the other parameters as described in the following procedure.

- Do not confuse xPC Target Scope blocks with standard Simulink Scope blocks.

- For more on using xPC Target Scope blocks, see "xPC Target Scope Usage" on page 5-25.

- For more on using target scopes, see "Target Scope Usage" on page 5-26.

- This procedure uses the model my_xpc_osc2 as an example. To access the example folder, type:

```
addpath (fullfile(matlabroot, 'help', 'toolbox', 'xpc',
      'examples'));
```

1 In the MATLAB window, type my_xpc_osc2.

The Simulink block diagram opens for the model my_xpc_osc2.

2 Double-click the block labeled Scope (xPC).

The Block Parameters: Scope (xPC) dialog box opens. By default, the target scope dialog box is displayed.

3 In the **Scope number** box, a unique number is displayed that identifies the scope. This number is incremented each time you add a new xPC Target Scope block.

This number identifies the xPC Target Scope block and the scope screen on the host or target computers.

4 From the **Scope type** list, select Target if it is not already selected. The updated dialog box is displayed.

5 Select the **Start scope when application starts** check box to start a scope when the target application is downloaded and started. The scope window opens automatically.

In Stand Alone mode, this setting is mandatory because the host computer is not available to issue a command to start scopes.

6 From the **Scope mode** list, select Numerical, Graphical redraw, Graphical sliding, or Graphical rolling.

If you have a scope type of Target and a scope mode of Numerical, the scope block dialog box adds a **Numerical format** box. You can define the display format for the data. If you choose not to complete the **Numerical format** box, the xPC Target software displays the signal using the default format of %15.6f, which is a floating-point format, without a label.

7 If you have selected scope mode Numerical, in the **Numerical format** box, type a label and associated numeric format type in which to display signals. By default, the entry format is floating-point without a label, %15.6f. The **Numerical format** box takes entries of the format:

'[LabelN] [%width.precision][type] [LabelX]'

- LabelN is the label for the signal. You can use a different label for each signal or the same label for each signal. This argument is optional.

- width is the minimum number of characters to offset from the left of the screen or label. This argument is optional.

- precision is the maximum number of decimal places for the signal value. This argument is optional.

- type is the data type for the signal format. You can use one or more of the following types.

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

- `LabelX` is a second label for the signal. You can use a different label for each signal or the same label for each signal. This argument is optional.

Enclose the contents of the **Numerical format** text box in single quotation marks.

For example:

```
'Foo %15.2f end'
```

For a whole integer signal value, enter `0` for the precision value. For example:

```
'Foo1 %15.0f end'
```

For a line with multiple entries, delimit each entry with a command and enclose the entire string in single quotation marks. For example:

```
'Foo2 %15.6f end,Foo3 %15.6f end2'
```

You can have multiple **Numerical format** entries, separated by a comma. If you enter one entry, that entry applies to each signal (scalar expansion). If you enter fewer label entries than signals, the first entry applies to the first signal, the second entry applies to the second signal, and so forth. The last entry is scalar expanded for the remaining signals. If you have two entries and one signal, the software ignores the second label entry and applies the first entry. You can enter as many format entries as you have signals for the scope. The format string has a maximum length of 100 characters, including spaces, for each signal.

8 Select the **Grid** check box to display grid lines on the scope. This parameter is applicable only for target scopes with scope modes of type `Graphical redraw`, `Graphical sliding`, or `Graphical rolling`.

9 In the **Y-Axis limits** box, enter a row vector with two elements. The first element is the lower limit of the *y*-axis and the second element is the upper limit. If you enter `0` for both elements, scaling is set to `auto`. This parameter is applicable only for target scopes with scope modes of type `Graphical redraw`, `Graphical sliding`, or `Graphical rolling`.

10 In the **Number of samples** box, enter the number of values to be acquired in a data package.

- If you select a **Scope mode** of `Graphical redraw`, the display redraws the graph every Number of samples.

- If you select a **Scope mode** of `Numerical`, the block updates the output every Number of samples.

- If you select a **Trigger mode** other than `FreeRun`, this parameter can specify the Number of samples to be acquired before the next trigger event.

11 In the **Number of pre/post samples** box, enter the number of samples to save or skip. To save N samples before a trigger event, specify the value `-N`. To skip N samples after a trigger event, specify the value N. The default is `0`.

12 In the **Decimation** box, enter a value to indicate that data must be collected at each sample time (1) or at less than every sample time (2 or greater).

13 From the **Trigger mode** list, select `FreeRun`.

- If you select `FreeRun` or `Software Triggering`, you do not need to enter additional parameters.

- If you select `Signal Triggering`, then enter the following additional parameters, as required:

  - In the **Trigger signal** box, enter the index of a signal previously added to the scope.

    This parameter does not apply if the **Add signal port to connect a signal trigger source** check box is selected.

  - (Alternatively) Click the **Add signal port to connect a signal trigger source** check box, then connect an arbitrary trigger signal to the port `Trigger signal`.

  - In the **Trigger level** box, enter a value for the signal to cross before triggering.

  - From the **Trigger slope** list, select one of `Either`, `Rising`, or `Falling`.

- If you select `Scope Triggering`, then enter the following additional parameters, as required:

- **−** In the **Trigger scope number** box, enter the scope number of a Scope block. If you use this trigger mode, you must also add a second Scope block to your Simulink model.

- **−** If you want the scope to trigger on a specific sample of the other scope, enter a value in the text box **Sample to trigger on (-1 for end of acquisition)**. The default value is 0, and indicates that the triggering scope and the triggered (current) scope start simultaneously.

  For more information on this field, see "Trigger One Scope with Another Scope" on page 7-21.

The target scope dialog box looks like this figure.

Sink Block Parameters: Scope (xPC)

xpcscopeblock (mask) (link)

xPC Target Scope
Configure scope to acquire signal data.
Scope can be of type target, host, or file.

Parameters

Scope number:

1

Scope type: Target

☑ Start scope when application starts

Scope mode: Graphical redraw

☑ Grid

Y-axis limits:

[0,0]

Number of samples:

1000

Number of pre/post samples:

0

Decimation:

1

Trigger mode: FreeRun

OK   Cancel   Help   Apply
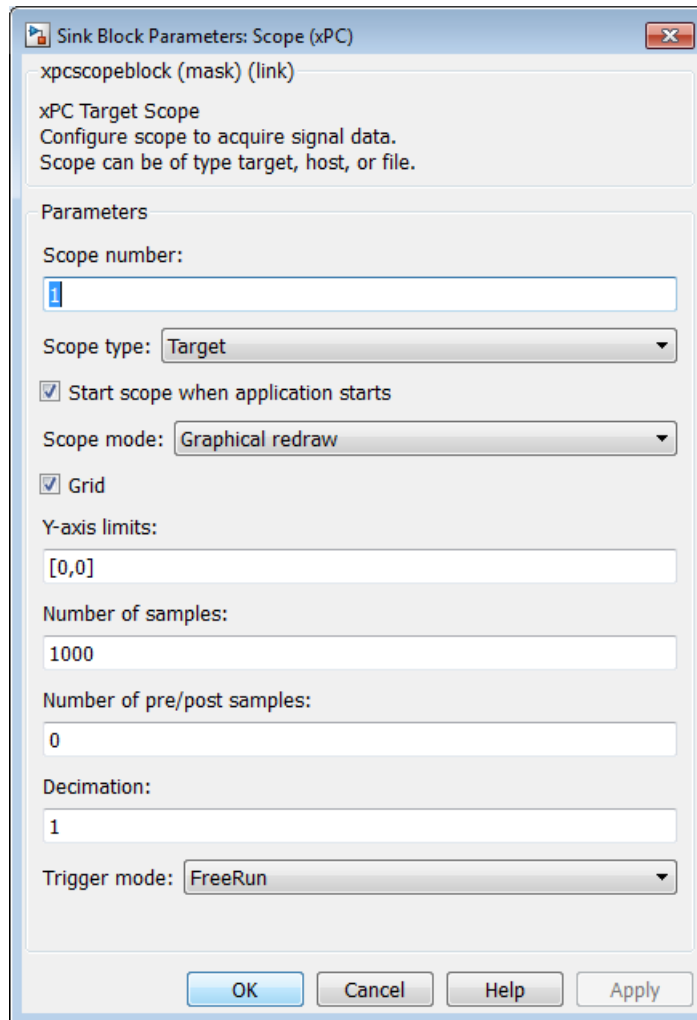
14 Click **OK**.

15 From the **File** menu, click **Save As**. The model is saved as my_xpc_osc2.

xPC Target Scope Usage

- To monitor an output signal from a Constant block by connecting it to an xPC Target Scope block, you must add a test point for the Constant block output signal.

- You can add an xPC Target scope only to the topmost model, not to a referenced model. To log signals from referenced models, use xPC Target Explorer scopes or xPC Target language scope objects.

- When the target application is built and downloaded, the xPC Target kernel creates a scope representing the Scope block. To change xPC Target Scope parameters after building the target application or while it is running, assign the scope to a MATLAB variable using the target object method `xpctarget.xpc.getscope`. If you use `xpctarget.xpc.getscope` to remove a scope created during the build and download process, and then restart the target application, the xPC Target kernel recreates the scope.

- If the output of a `Mux` block is connected to the input of an xPC Target Scope block, the signal might not be observable. To observe the signal, add a unity gain block (a `Gain` block with a gain of 1) between the `Mux` block and the xPC Target Scope block. See "Nonobservable Signals and Parameters" on page 5-140.

Target Scope Usage

- xPC Target supports ten target scopes. Each target scope can contain up to 10 signals.

- For a target scope, logged data (`sc.Data` and `sc.Time`) is not accessible over the command-line interface on the host computer. This is because logged data is only accessible when the scope object status (`sc.Status`) is set to `Finished`. When the scope completes one data cycle (time to collect the number of samples), the scope engine automatically restarts the scope instead of setting`sc.Status` to `Finished`.

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

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

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

Configure Host Scope (xPC) Blocks

xPC Target includes a specialized Scope (xPC) block that you can configure to display signal and time data on the host computer monitor. To do this, add a Scope (xPC) block to the model, select **Scope type** to Host and configure the other parameters as described in the following procedure.

- Do not confuse xPC Target Scope blocks with standard Simulink Scope blocks.

- For more on using xPC Target Scope blocks, see "xPC Target Scope Usage" on page 5-25.

- For more on host scopes, see "Host Scope Usage" on page 5-30.

- This procedure uses the model my_xpc_osc2 as an example. To access the example folder, type:

  ```
  addpath (fullfile(matlabroot, 'help', 'toolbox', 'xpc',
      'examples'));
  ```

1 In the MATLAB window, type my_xpc_osc2.

The Simulink block diagram opens for the model my_xpc_osc2.

2 Double-click the block labeled Scope (xPC).

The Block Parameters: Scope (xPC) dialog box opens. By default, the target scope dialog box is displayed.

3 In the **Scope number** box, a unique number is displayed that identifies the scope. This number is incremented each time that you add a new xPC Target scope.

This number identifies the xPC Target Scope block and the scope screen on the host or target computers.

4 From the **Scope type** list, select Host. The updated dialog box is displayed.

5 Select the **Start scope when application starts** check box to start a scope when the target application is downloaded and started. With a target scope, the scope window opens automatically. With a host scope, you can open a host scope viewer window from xPC Target Explorer.

In Stand Alone mode, this setting is mandatory because the host computer is not available to issue a command to start scopes.

6 In the **Number of samples** box, enter the number of values to be acquired in a data package.

7 In the **Number of pre/post samples** box, enter the number of samples to save or skip. To save N samples before a trigger event, specify the value N. To skip N samples after a trigger event, specify the value N. The default is 0.

8 In the **Decimation** box, enter a value to indicate that data must be collected at each sample time (1) or at less than every sample time (2 or greater).

9 From the **Trigger mode** list, select FreeRun.

- If you select FreeRun or Software Triggering, you do not need to enter additional parameters.

- If you select Signal Triggering, then enter the following additional parameters, as required:

  - In the **Trigger signal** box, enter the index of a signal previously added to the scope.

    This parameter does not apply if the **Add signal port to connect a signal trigger source** check box is selected.

  - (Alternatively) Click the **Add signal port to connect a signal trigger source** check box, then connect an arbitrary trigger signal to the port Trigger signal.

  - In the **Trigger level** box, enter a value for the signal to cross before triggering.

  - From the **Trigger slope** list, select one of Either, Rising, or Falling.

- If you select Scope Triggering, then enter the following additional parameters, as required:

  - In the **Trigger scope number** box, enter the scope number of a Scope block. If you use this trigger mode, you must also add a second Scope block to your Simulink model.

  - If you want the scope to trigger on a specific sample of the other scope, enter a value in the text box **Sample to trigger on (-1 for end of**

acquisition). The default value is `0`, and indicates that the triggering scope and the triggered (current) scope start simultaneously.

For more information on this field, see "Trigger One Scope with Another Scope" on page 7-21.

The host scope dialog box looks like this figure.
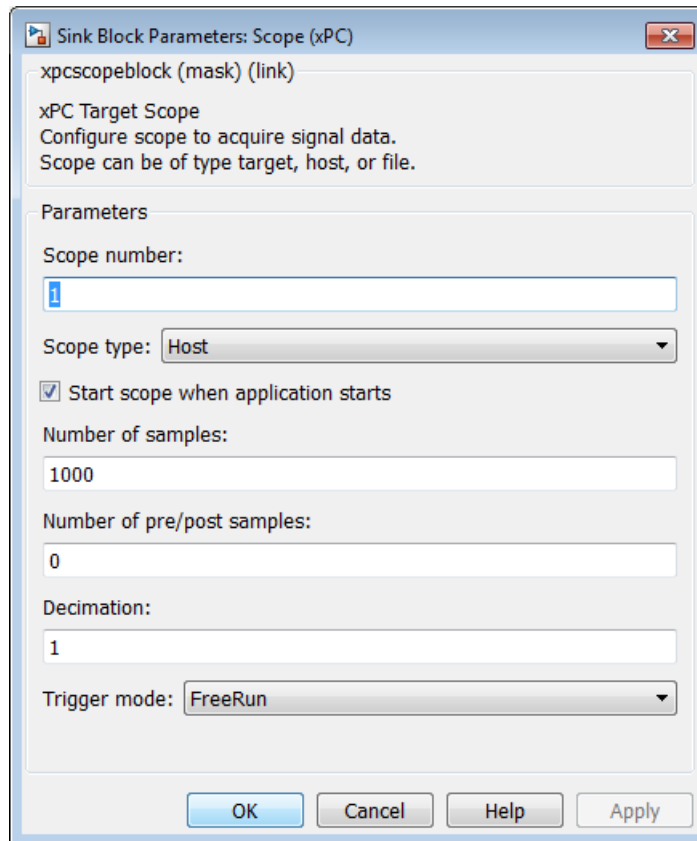
10 Click **OK**.

11 From the **File** menu, click **Save As**. The model is saved as `my_xpc_osc2`.

Host Scope Usage

- xPC Target supports as many host scopes as the target computer resources can support. Each host scope can contain as many signals as the target computer has resources to support them.

- Use host scopes to log signal data triggered by an event while your target application is running. The host 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 that you entered in the **Number of samples** parameter.

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

Create Target Scopes Using xPC Target Explorer

You can create a virtual target scope on the target computer using xPC Target Explorer. These scopes have the full capabilities of the Scope (xPC) block in `Target` mode, but do not persist past the current execution.

Note For information on using target scope blocks, see "Configure Target Scope (xPC) Blocks" on page 5-19 and "Target Scope Usage" on page 5-26.

This procedure uses the model `xpcosc` as an example. You must have already completed the following setup:

1 Built and downloaded the target application to the target computer using Simulink (⬛ on the toolbar).

2 Run xPC Target Explorer (command `xpcexplr`).

3 Connected to the target computer in the **Targets** pane (🔵 on the toolbar).

4 Set property **Stop time** to `inf` in the **Applications** pane (⚙ on the toolbar).

To configure a virtual target scope:

1 In the **Scopes** pane, expand the **xpcosc** node.

2 To add a target scope, select **Target Scopes** and then click the Add Scope icon 🟢 on the toolbar.

The new scope appears under node **Target Scopes**, for example **Scope 1**.

3 Select **Scope 1** and then click the Properties icon 🟦 on the toolbar.

4 In the **Scope Properties** workspace, click **Signals**. You add signals from the **Target Applications** Signals workspace.

5 In the **Target Applications** pane, expand the target application node and then node **Model Hierarchy**.

6 Select the model node and then click the View Signals icon ⊡ on the toolbar.

The Signals workspace opens, showing a table of signals with properties and actions.

7 In the Signals workspace, to add signal Signal Generator to **Scope1**, click the down arrow next to the Scopes icon 🗾 in its Actions column.

A list of scope types appears. **Scope 1**appears under node **Target Scopes**.

8 Click the Add Signal(s) icon ⊕ next to **Scope1** under node **Target Scopes**.

9 Add signal Integrator1 to **Scope 1** in the same way.

The dialog box looks like this figure.
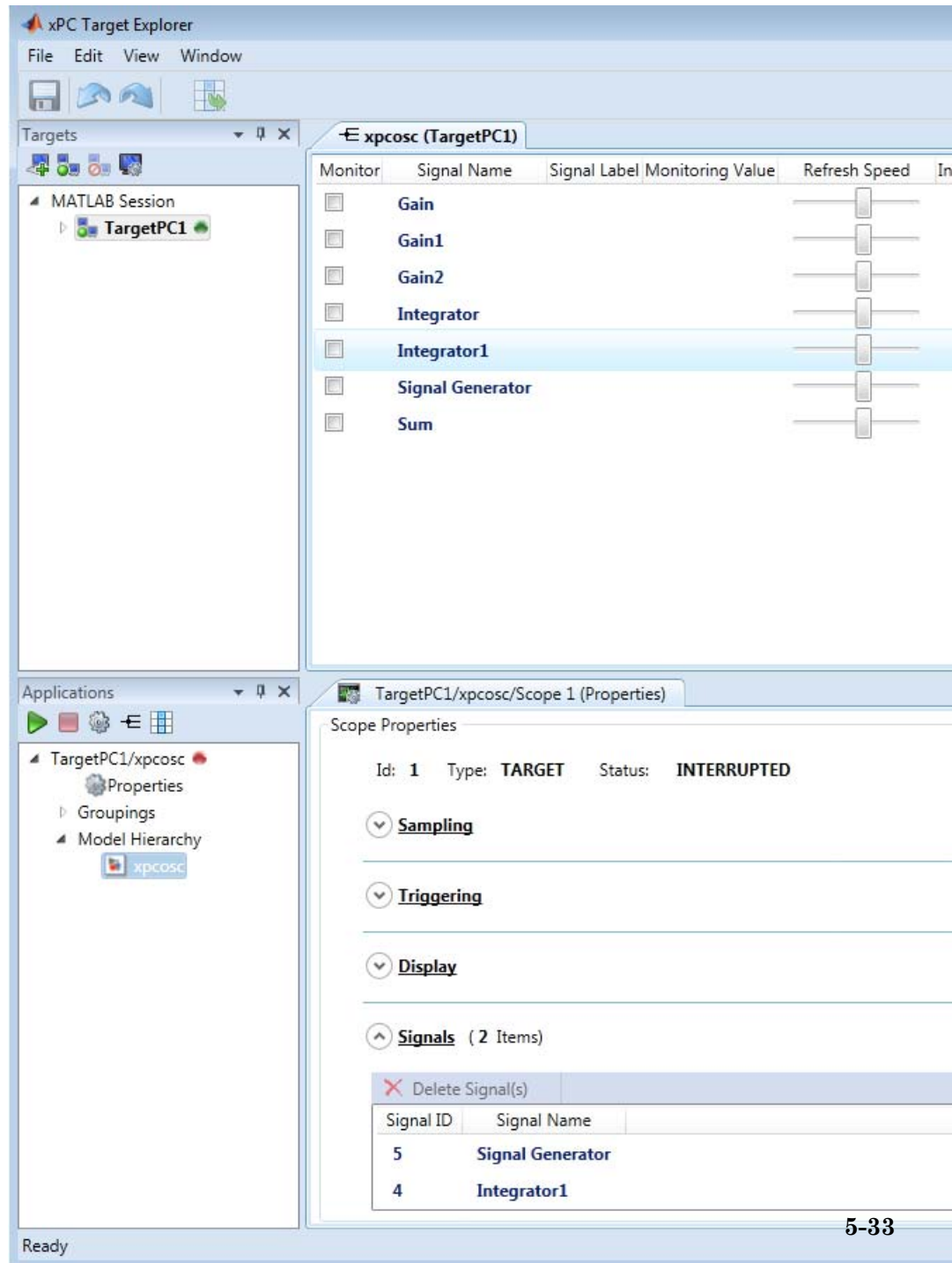
10 To start execution, click the target application and then click the Start icon ▶ on the toolbar.

The application starts running. No output appears on the target computer monitor.

11 To start **Scope 1**, click **Scope 1** in the **Scopes** pane and then click the Start Scope icon 🔳 on the toolbar.

Output for signals Signal Generator and Integrator1 appears on the target computer monitor.

The target computer screen looks like this figure.
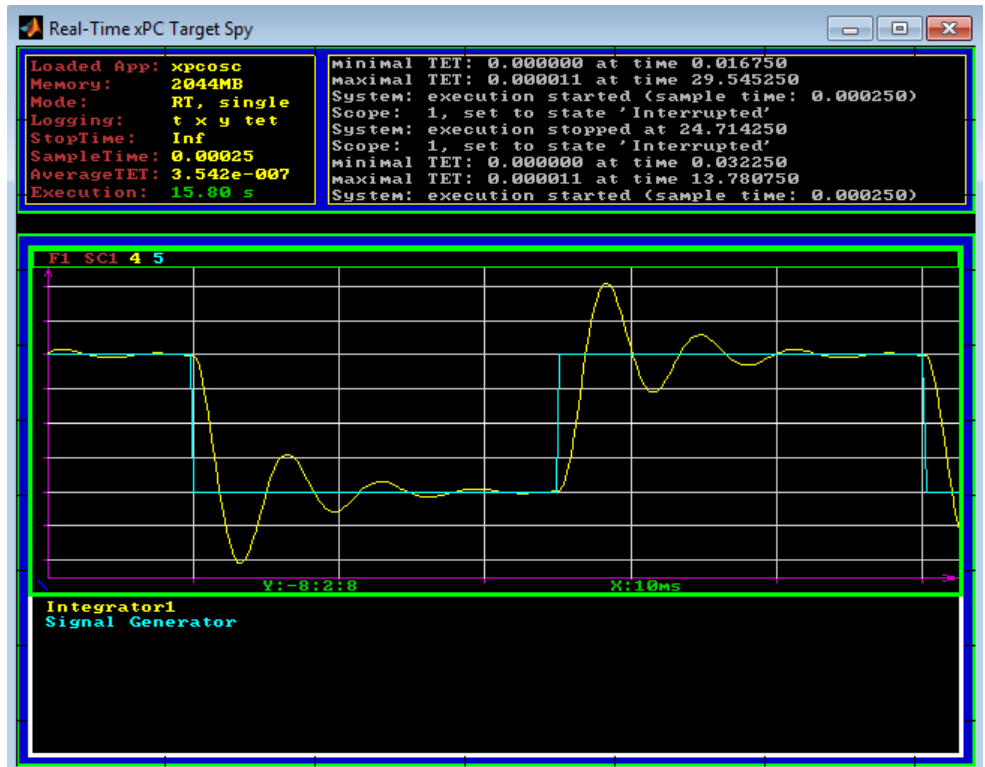
12 To stop **Scope 1**, click **Scope 1** in the **Scopes** pane and then click the Stop Scope icon ![icon] on the toolbar.

The signals shown on the target computer stop updating while the target application continues running. The target computer monitor displays a message like this message:

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

13 To stop execution, click the target application and then click the Stop icon ![icon] on the toolbar.

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

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

- You can create a virtual target scope from the scope types list by clicking **Add Scope** next to scope type **Target Scopes**.

- You can add or remove signals from a virtual target scope while the scope is either stopped or running.

- To group signals, see "Create Signal Groups Using xPC Target Explorer" on page 5-54.

- To configure the target computer display, see "Configure Target Scopes Using xPC Target Explorer" on page 5-50.

- To configure data sampling, see "Configure Scope Sampling Using xPC Target Explorer" on page 5-37.

- To configure scope triggering, see "Trigger Scopes Interactively Using xPC Target Explorer" on page 5-40 and "Trigger Scopes Noninteractively Using xPC Target Explorer" on page 5-44.

- To make both workspaces visible at the same time, click and hold the tab for one workspace and drag it down until the following icon appears in the middle of the dialog box: ![icon]. Continue to drag until the cursor reaches the required quadrant, and then release the mouse button.

- If a block is unnamed, xPC Target Explorer does not display signals or a node for that block. To reference such a block, provide an alphanumeric

name for that block, rebuild and download the model to the target computer, and reconnect the MATLAB session to the target computer.

Configure Scope Sampling Using xPC Target Explorer

You can customize sampling for xPC Target scopes to facilitate data access to the running model. You can configure sampling whether you added a Scope (xPC) block to the model or added the scope at run time.

This procedure uses the model xpcosc as an example. You must have already completed the procedure in "Create Target Scopes Using xPC Target Explorer" on page 5-31. Target execution and scopes must be stopped.

1 Select **Scope 1** and open the Properties pane (🖼️ on the **Scopes** toolbar).

2 In the **Scope 1** properties pane, click **Sampling**.

3 In the **Number of Samples** box, enter the number of values to be acquired in a data package, here 250.

   If you select a **Display mode** of Graphical redraw, the display redraws the graph every Number of Samples.

   If you select a **Display mode** of Numerical, the block updates the output every Number of Samples.

   If you select a **Trigger Mode** other than FreeRun, this parameter can specify the number of samples to be acquired before the next trigger event. See "Trigger Scopes Interactively Using xPC Target Explorer" on page 5-40 and "Trigger Scopes Noninteractively Using xPC Target Explorer" on page 5-44.

4 In the **Decimation** box, enter 10 to indicate that data must be collected at every 10th sample time. The default is 1, to collect data at every sample time.

5 In the **Number of pre/post samples** box, enter the number of samples to save or skip. To save N samples before a trigger event, specify the value   N. To skip N samples after a trigger event, specify the value N. The default is 0.

   The dialog box looks like this figure.
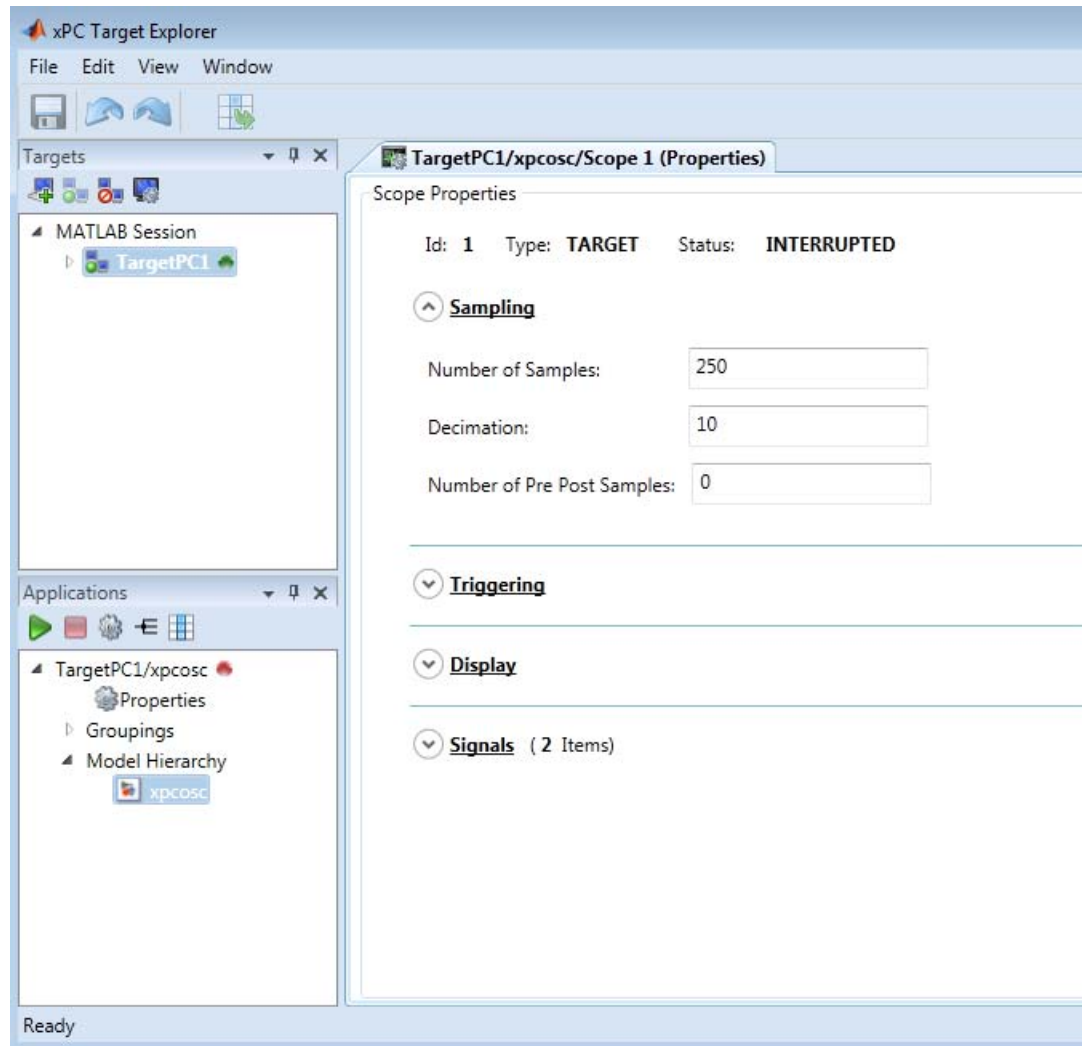
6 To see the effect of these settings, start execution (▶ on the **Applications** toolbar).

7 Start **Scope 1** (⬛ on the toolbar).

Output for signals Signal Generator and Integrator1 appears on the target computer monitor.
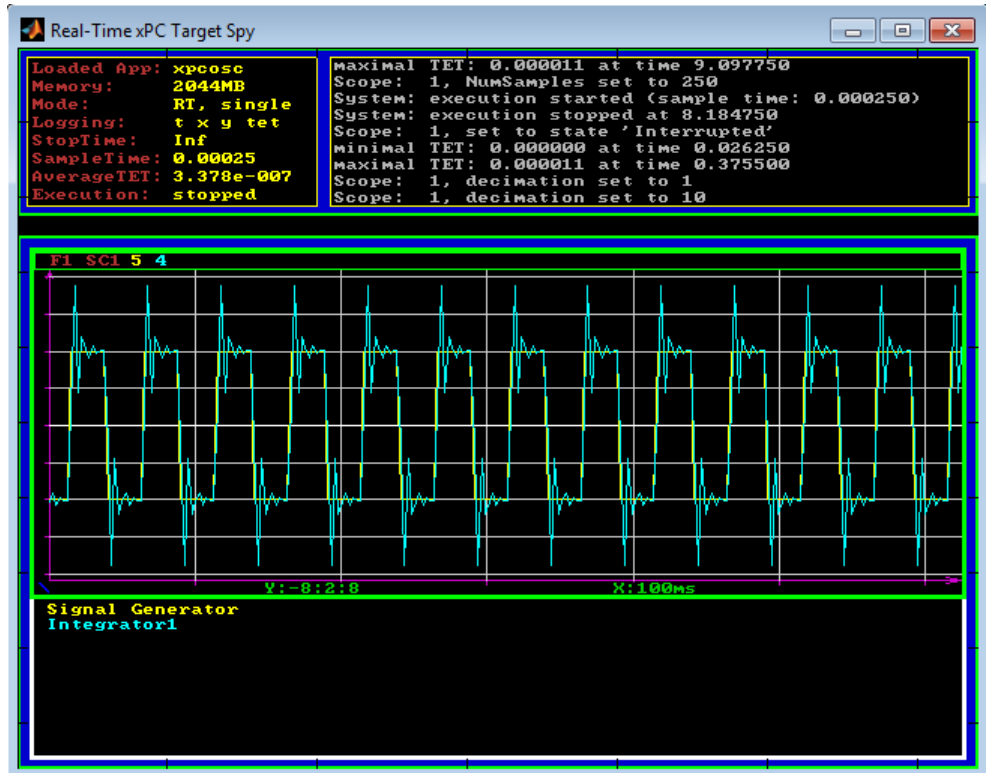
8 Stop **Scope 1** (on the toolbar).

9 Stop execution (on the **Applications** toolbar).

Trigger Scopes Interactively Using xPC Target Explorer

You can customize scope triggering for xPC Target scopes to facilitate your interaction with the running model. You can configure triggering whether you created the scope by adding a Scope (xPC) block to the model or by adding the scope at run time.

This procedure uses the model xpcosc as an example. You must have already completed the procedure in "Create Target Scopes Using xPC Target Explorer" on page 5-31. Target execution and scopes must be stopped.

1 Start execution (on the **Applications** toolbar).

2 Select **Scope 1** and open the Properties pane (on the **Scopes** toolbar).

3 In the **Scope 1** pane, click **Triggering**.

4 Select **Trigger Mode** Freerun.

By default, the **Trigger Mode** is set to Freerun. In this mode, the scope triggers automatically as soon as it is started. It displays data until it is stopped.

5 Start and stop **Scope 1** (and on the toolbar).

Signal data is displayed on the target computer monitor when the scope starts and stops when the scope stops.

6 Select **Trigger Mode** Software.

In this mode, the scope triggers when you select **Scope 1** and then click the Trigger icon on the toolbar. It runs until you click it again.

7 Start **Scope 1** (on the toolbar).

The Trigger icon is enabled on the toolbar.

8 Click the Trigger icon on the **Scopes** toolbar.

The current signal data is displayed on the target computer monitor when you click the icon.

9 Stop **Scope 1** (on the toolbar).

10 Select **Trigger Mode** Scope.

Settings **Trigger scope** and **Trigger scope sample** appear.

11 Set **Trigger scope** to 1. Press **Enter**.

The current signal data is displayed when you click the Trigger icon on the toolbar.

12 Leave **Trigger scope sample** set to 0.

Scope 1 triggers on the first sample after you click the Trigger icon.

13 Start **Scope 1** (on the toolbar).

The Trigger icon is enabled on the toolbar.

14 Click the Trigger icon on the **Scopes** toolbar.

The current signal data is displayed on the target computer monitor when you click the icon.

15 Stop **Scope 1** (on the toolbar).

The dialog box looks like this figure.
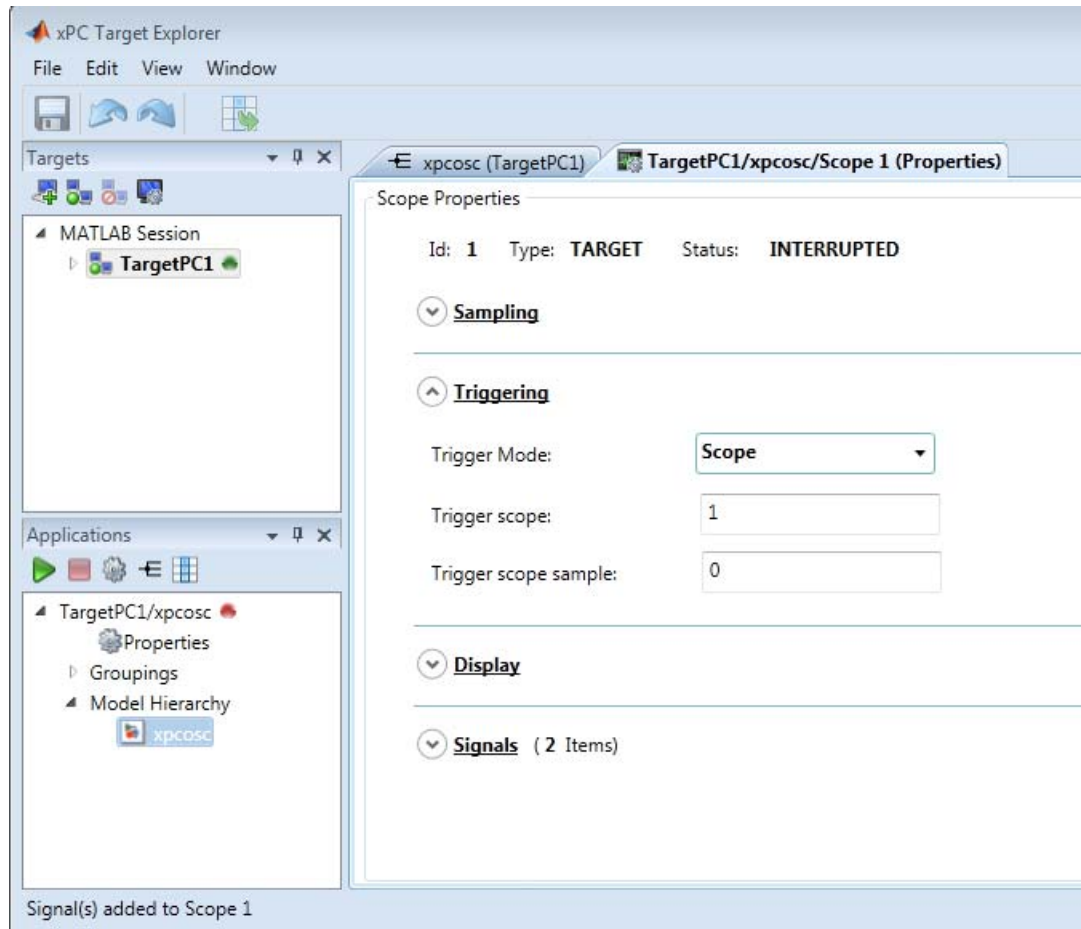
The target monitor looks like this figure.
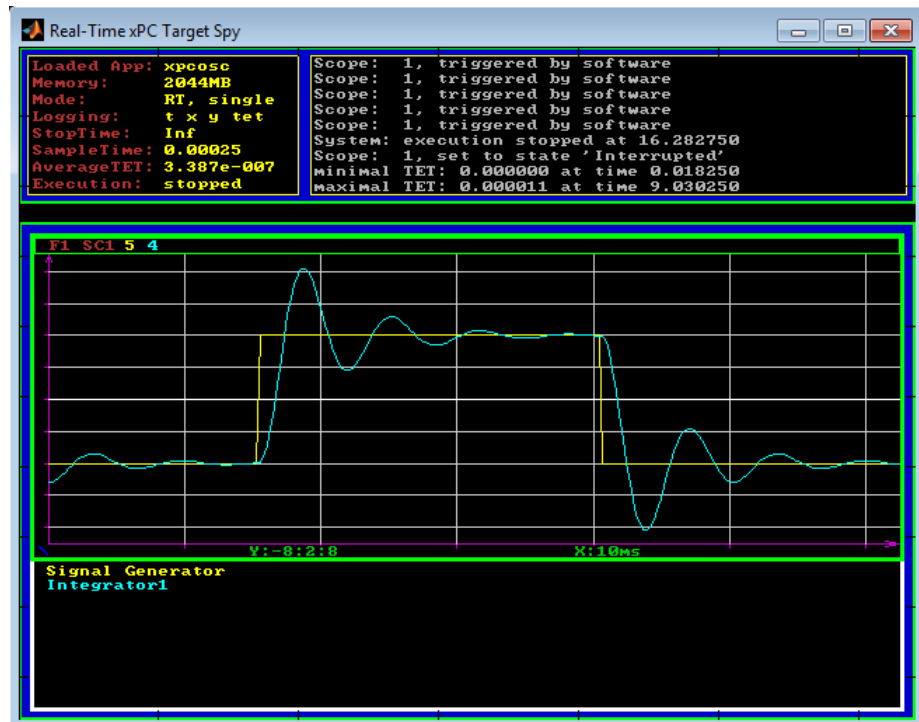
16 Stop execution (■ on the **Applications** toolbar).

Trigger Scopes Noninteractively Using xPC Target Explorer

You can customize scope triggering for xPC Target scopes to facilitate your control of the running model. You can configure triggering whether you added a Scope (xPC) block to the model or added the scope at run time.

This procedure uses the model xpcosc as an example. You must have already completed the procedure in "Create Target Scopes Using xPC Target Explorer" on page 5-31. Target execution and scopes must be stopped.

1 Start execution (▶ on the **Applications** toolbar).

2 Select **Scope 1** and open the Properties pane (🖼 on the **Scopes** toolbar).

3 In the **Scope 1** pane, click **Triggering**.

4 Select **Trigger Mode** Signal.

   Settings **Trigger Signal**, **Trigger Slope**, and **Trigger Level** appear.

5 Set **Trigger Signal** to Signal Generator.

6 Set **Trigger Slope** to Rising.

7 Leave **Trigger Level** as 0, indicating that the signal crosses 0 before **Scope 1** triggers.
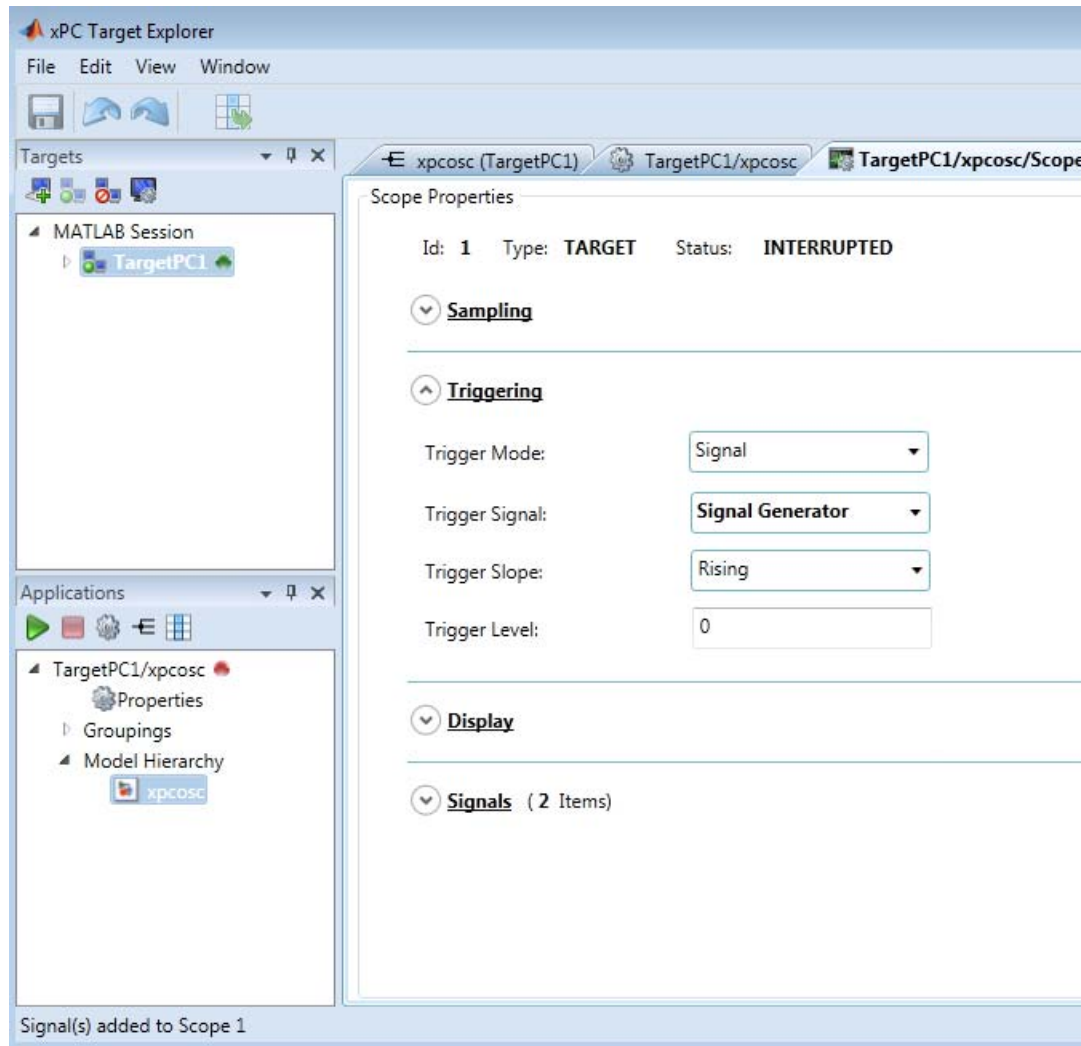
8 Start **Scope 1** (on the toolbar).

Signal data is displayed on the target computer monitor, with the rising pulse of Signal Generator just beyond the left side.
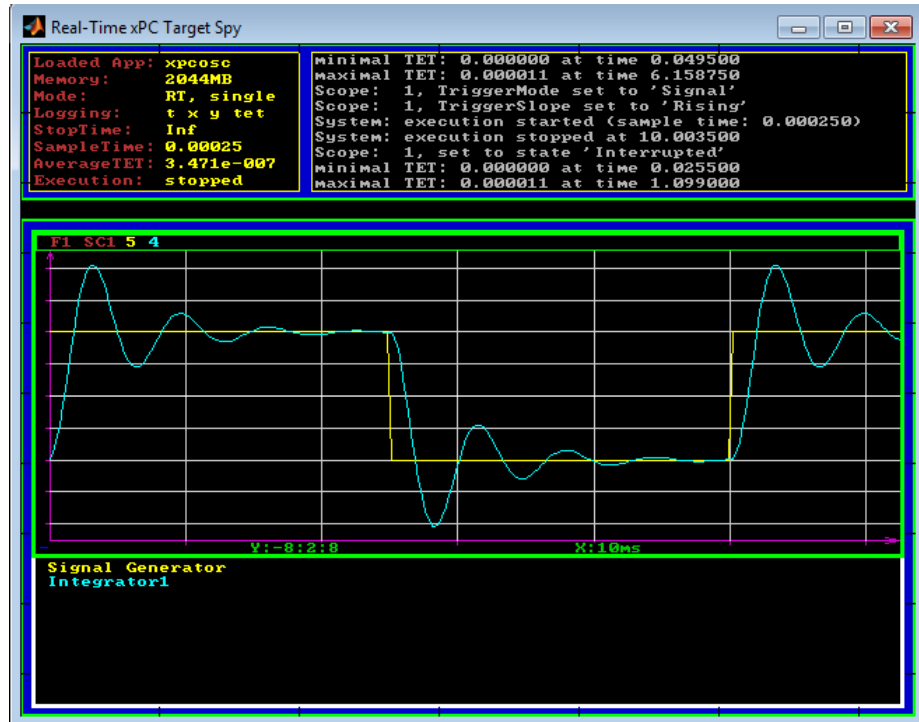
9 Stop **Scope 1** (🔲 on the toolbar).

10 Add target scope **Scope 2** (🔲 on the **Scopes** toolbar).

11 Open the Signals pane (🔲 on the **Applications** toolbar).

12 Add signal `Integrator` to **Scope 2** in the **Signals** pane.

13 In the **Scope 2** pane, click **Triggering**.

14 Select **Trigger Mode** Scope.

Settings **Trigger scope** and **Trigger scope sample** appear.

15 Set **Trigger scope** to 1. Press **Enter**. **Scope 2** then triggers when **Scope 1** triggers.

16 Leave **Trigger scope sample** set to 0. **Scope 2** triggers on the same
sample as **Scope 1**.
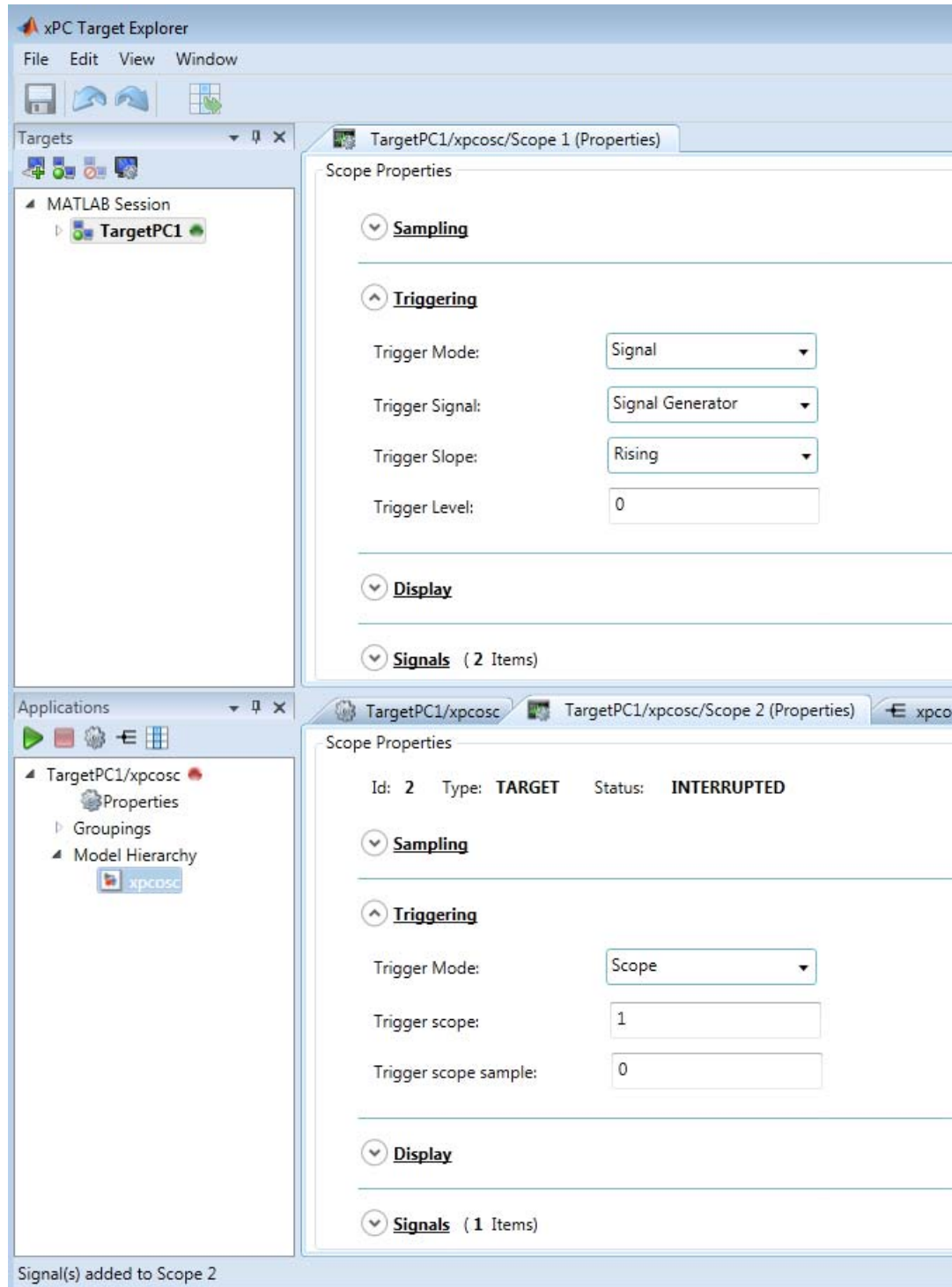
17 Start both **Scope 1** and **Scope 2** (![icon] on the toolbar). You must explicitly start and stop both scopes.

Scope 1 and **Scope 2** display signal data on the target computer monitor.
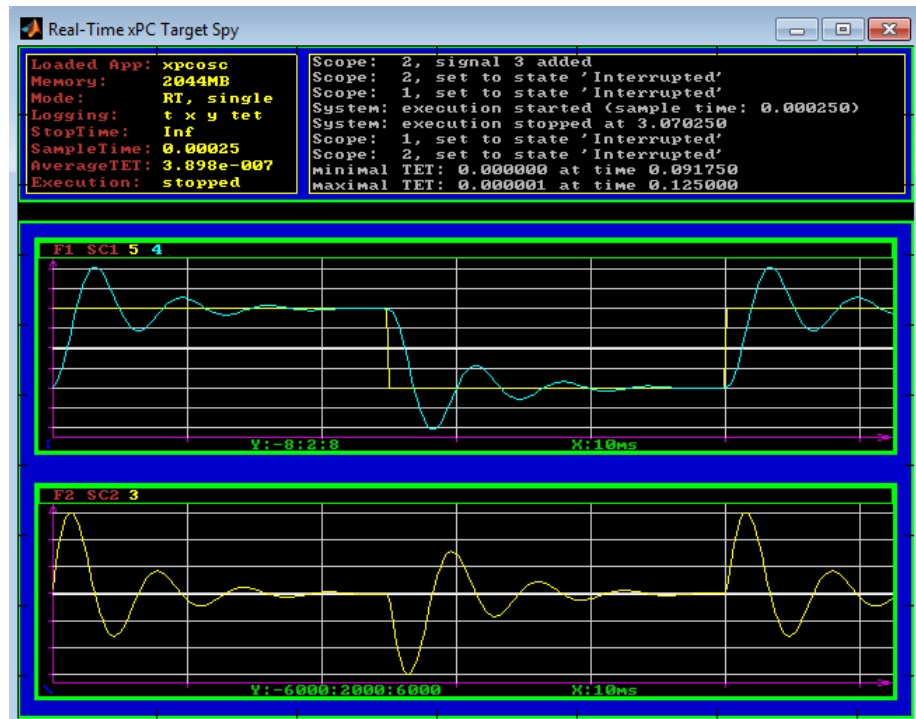
18 Stop both **Scope 1** and **Scope 2** (![icon] on the toolbar).

19 Stop execution (![icon] on the **Applications** toolbar).

Configure Target Scopes Using xPC Target Explorer

You can configure the target scope display to facilitate your view of the signal data. You can configure the display whether you added a Scope (xPC) block to the model or added the scope at run time.

This procedure uses the model xpcosc as an example. You must have already completed the procedure in "Create Target Scopes Using xPC Target Explorer" on page 5-31. Target execution and scopes must be stopped.

1 Start execution (on the **Applications** toolbar).

2 Select **Scope 1** and open the Properties pane (on the **Scopes** toolbar).

3 In the **Scope 1** pane, click **Display**.

4 Select **Display mode** Redraw and then click in the **Y-Limits** box.

This value is the default. It causes the scope display to redraw as soon as it has acquired as many samples as specified in **Number of Samples**.

5 Start **Scope 1** (on the toolbar).

Signal data is displayed on the target computer monitor, appearing to move to the left.

6 Enter [0,10] in the **Y-Limits** box and then press **Enter**. The default setting is [0,0], which automatically scales the output according to the signal values.

The display changes to show only values at and above the zero line.

7 Clear the **Grid (On/Off)** check box. By default, the box is selected.
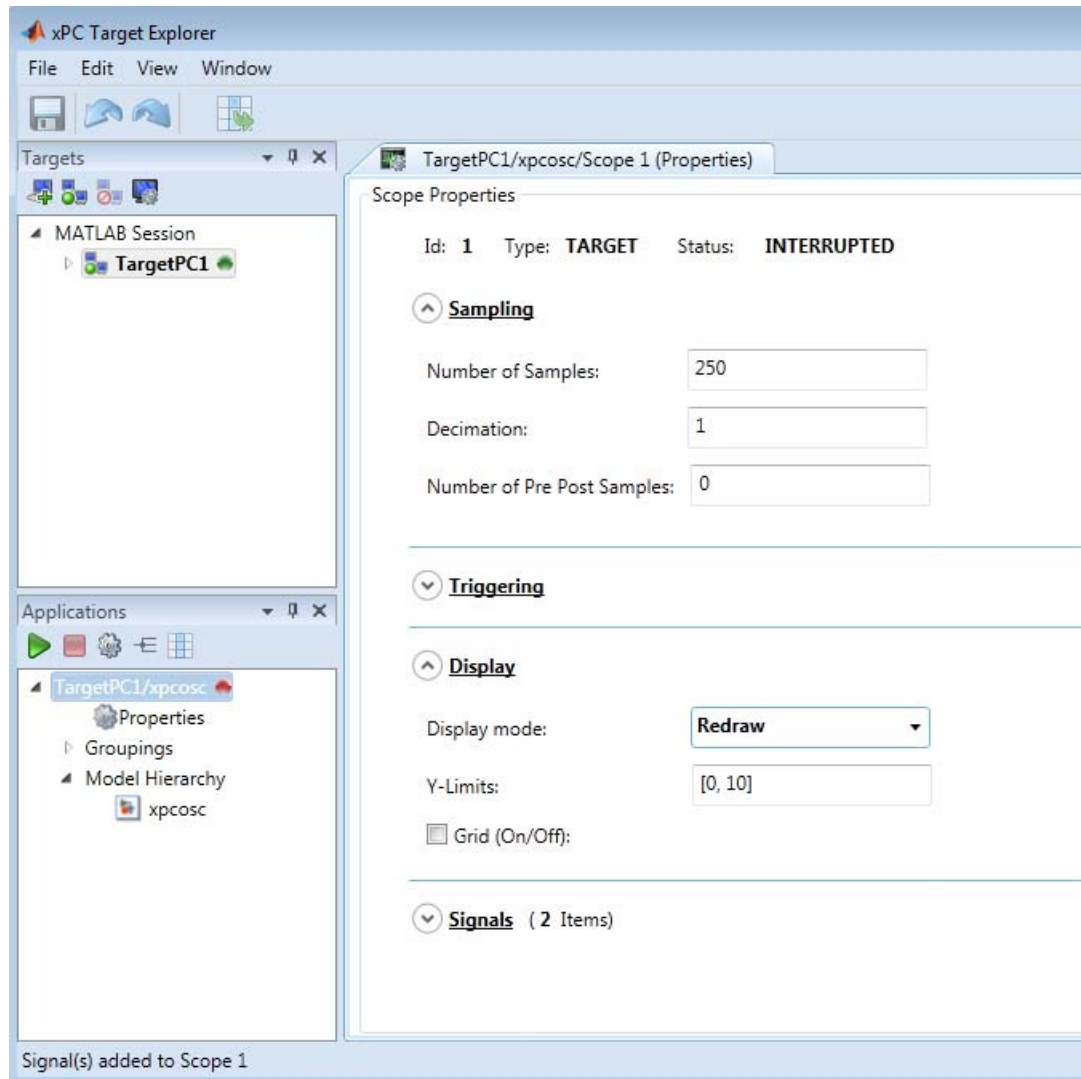
The target computer monitor looks like this figure.
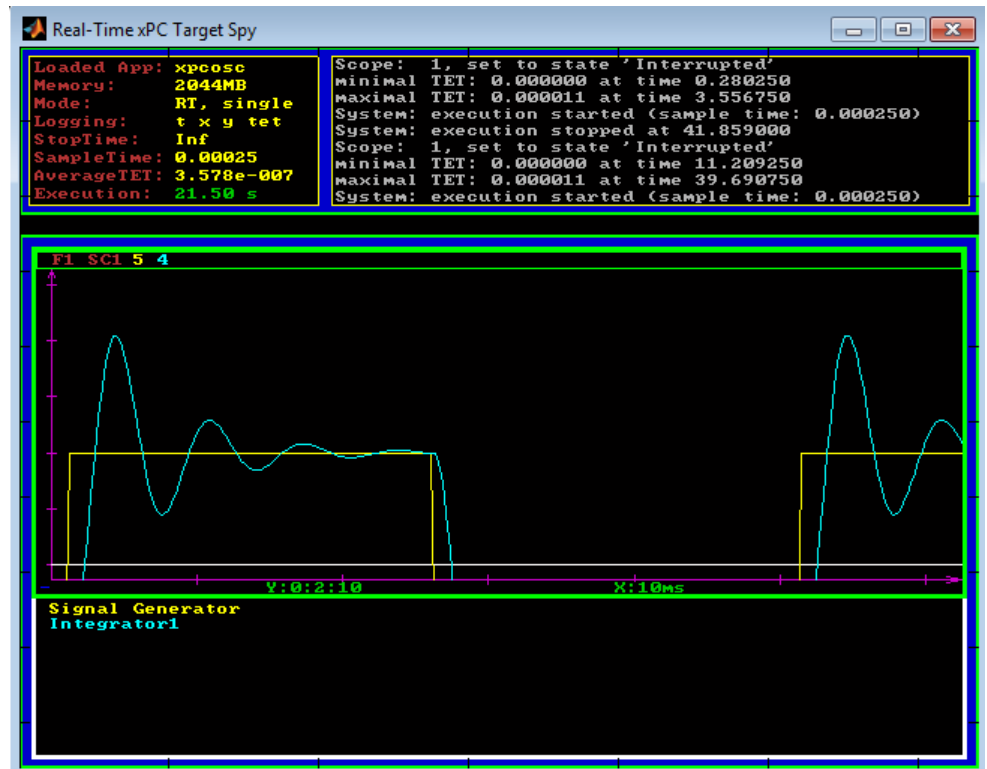
8 Select **Display mode** Numerical and then click in the **Y-Limits** box.

The grid and axes disappear. The target computer monitor displays the signals, color coded, in the default format of %15.6f (a floating-point format without a label).

9 Select **Display mode** Rolling and then click in the **Y-Limits** box.

The display changes to a display that continuously moves a window along the signal stream. New data enters the display from the right and then moves toward the left.

10 Select **Display mode** Sliding and then click in the **Y-Limits** box. In this mode, the scope refreshes continuously. New data overwrites the display from the left toward the right.

11 Stop **Scope 1** (on the toolbar).

12 Stop execution (on the **Applications** toolbar).

Create Signal Groups Using xPC Target Explorer

When testing a complex model with many signals, you frequently must select signals for tracing or monitoring from multiple parts and levels of the model hierarchy. You can make this task easier by using xPC Target Explorer to create a signal group and save it to disk.

This procedure uses the model xpcosc as an example. You must have already completed the following setup:

1 Built and downloaded the target application to the target computer using Simulink (![icon] on the toolbar).

2 Run xPC Target Explorer (command xpcexplr).

3 Connected to the target computer in the **Targets** pane (![icon] on the toolbar).

To create a signal group:

1 In the **Target Applications** pane, expand the target application node and right-click node **Groupings**.

2 Click **New Signal Group**.

The Add New Signal Group Item dialog box appears.

3 In the Add New Signal Group Item dialog box, enter a name in the **Name** text box, for example **signalgroup1.sig**. In the **Location** text box, enter a folder for the group file.

4 Click **OK**.

A new signal group appears, along with its Signal Group workspace.

5 In the **Target Applications** pane, expand the target application node and then expand node **Model Hierarchy**.

6 Select the model node and then click the View Signals icon ![icon] on the toolbar.

The Signals workspace opens, showing a table of signals with properties and actions.

7 In the Signal Groups workspace, to add signal `Signal Generator` to **signalgroup1.sig**, click the down arrow next to the Signals Grouping icon ⍒ in its Actions column.

A list of signal groups appears, including **signalgroup1.sig**.

8 Click the Add Signal(s) icon ⊕ next to **signalgroup1.sig**.

9 Add signal `Integrator1` to **signalgroup1.sig** in the same way.

10 Click in the Signal Group workspace, and then click the Save icon 💾 on the toolbar.

When you are monitoring a signal group, you can change the output format of the group by selecting one of the options in the **Format** column.
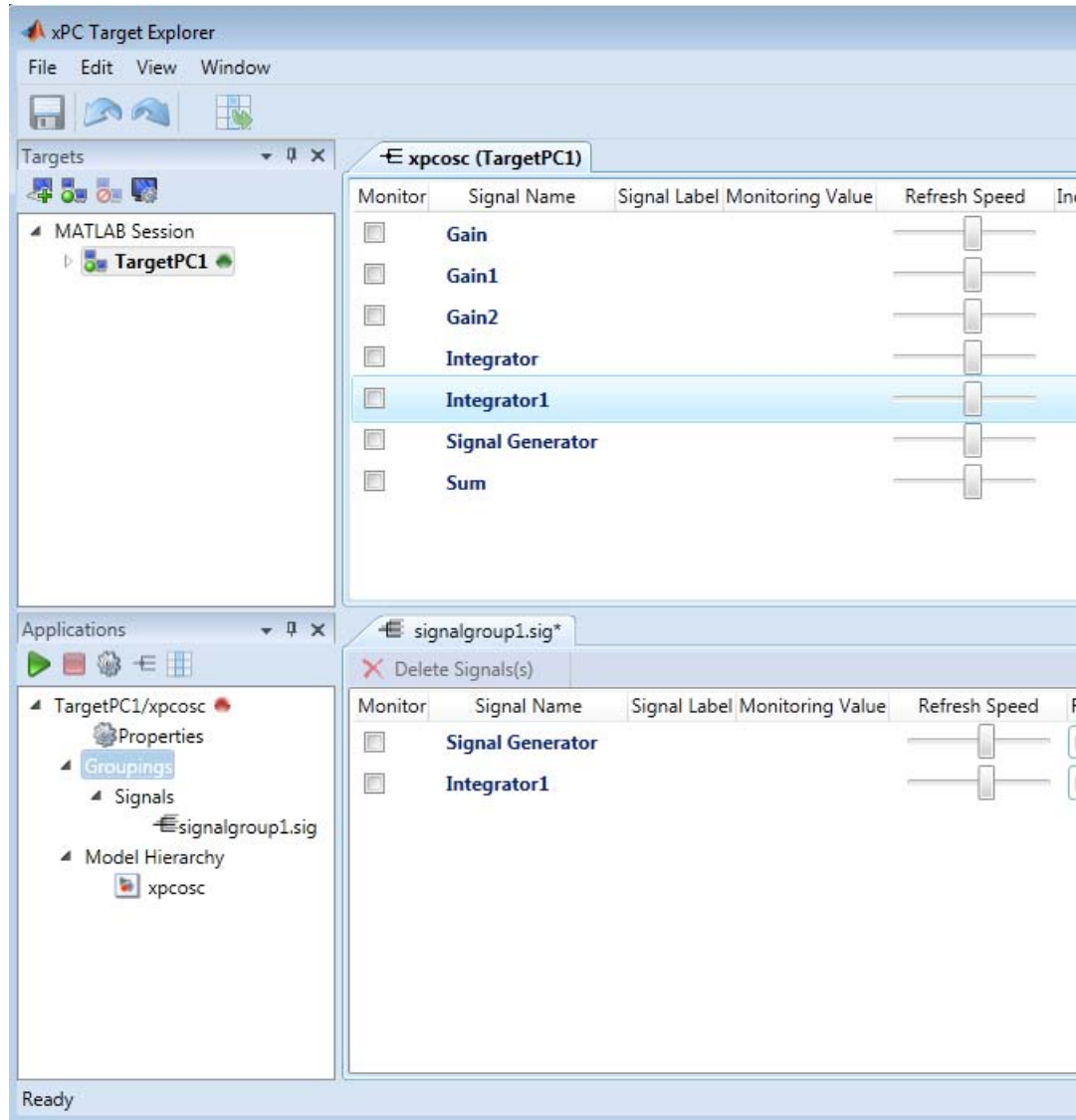
- To make both workspaces visible at the same time, click and hold the tab
  for one workspace and drag it down until the following icon appears in the

middle of the dialog box: . Continue to drag until the cursor reaches the required quadrant, and then release the mouse button.

- For more on monitoring individual signals in the group, see "Monitor Signals Using xPC Target Explorer" on page 5-5.

- For more on tracing individual signals using a target scope, see "Create Target Scopes Using xPC Target Explorer" on page 5-31.

- For more on tracing individual signals using a host scope, see "Create Host Scopes Using xPC Target Explorer" on page 5-58.

- For more on logging individual signals using a file scope, see "Create File Scopes Using xPC Target Explorer" on page 5-85.

Create Host Scopes Using xPC Target Explorer

You can create a virtual host scope on the target computer using xPC Target Explorer. These scopes have the full capabilities of the Scope (xPC) block in Host mode, but do not persist past the current execution.

Note For information on using host scope blocks, see "Configure Host Scope (xPC) Blocks" on page 5-27 and "Host Scope Usage" on page 5-30.

This procedure uses the model xpcosc as an example. You must have already completed the following setup:

1 Built and downloaded the target application to the target computer using

Simulink (on the toolbar).

2 Run xPC Target Explorer (command xpcexplr).

3 Connected to the target computer in the **Targets** pane (on the toolbar).

4 Set property **Stop time** to inf in the **Applications** pane (on the toolbar).

To configure a virtual host scope:

1 In the **Scopes** pane, expand the **xpcosc** node.

2 To add a host scope, select **Host Scopes** and then click the Add Scope icon on the toolbar.

Under the **Host Scopes** node, the new scope is displayed, for example **Scope 1**.

3 Expand **Scope 1** and then click the Properties icon on the toolbar.

4 In the **Scope Properties** pane, click **Signals**. Add signals from the **Target Applications** Signals workspace.

5 In the **Target Applications** pane, expand the target application node and then node **Model Hierarchy**.

6 Select the model node and then click the View Signals icon ⊨ on the toolbar.

7 In the Signals workspace, to add signal `Signal Generator` to **Scope1**, click the down arrow next to the Scopes icon 🖼 in its Actions column.

A list of scope types is displayed. **Scope 1** appears under node **Host Scopes**.

8 Click the Add Signal(s) icon ⊕ next to **Scope1** under node **Host Scopes**.

You can add or remove signals from a virtual host scope while the scope is either stopped or running.

9 Add signal `Integrator1` to **Scope 1** in the same way.
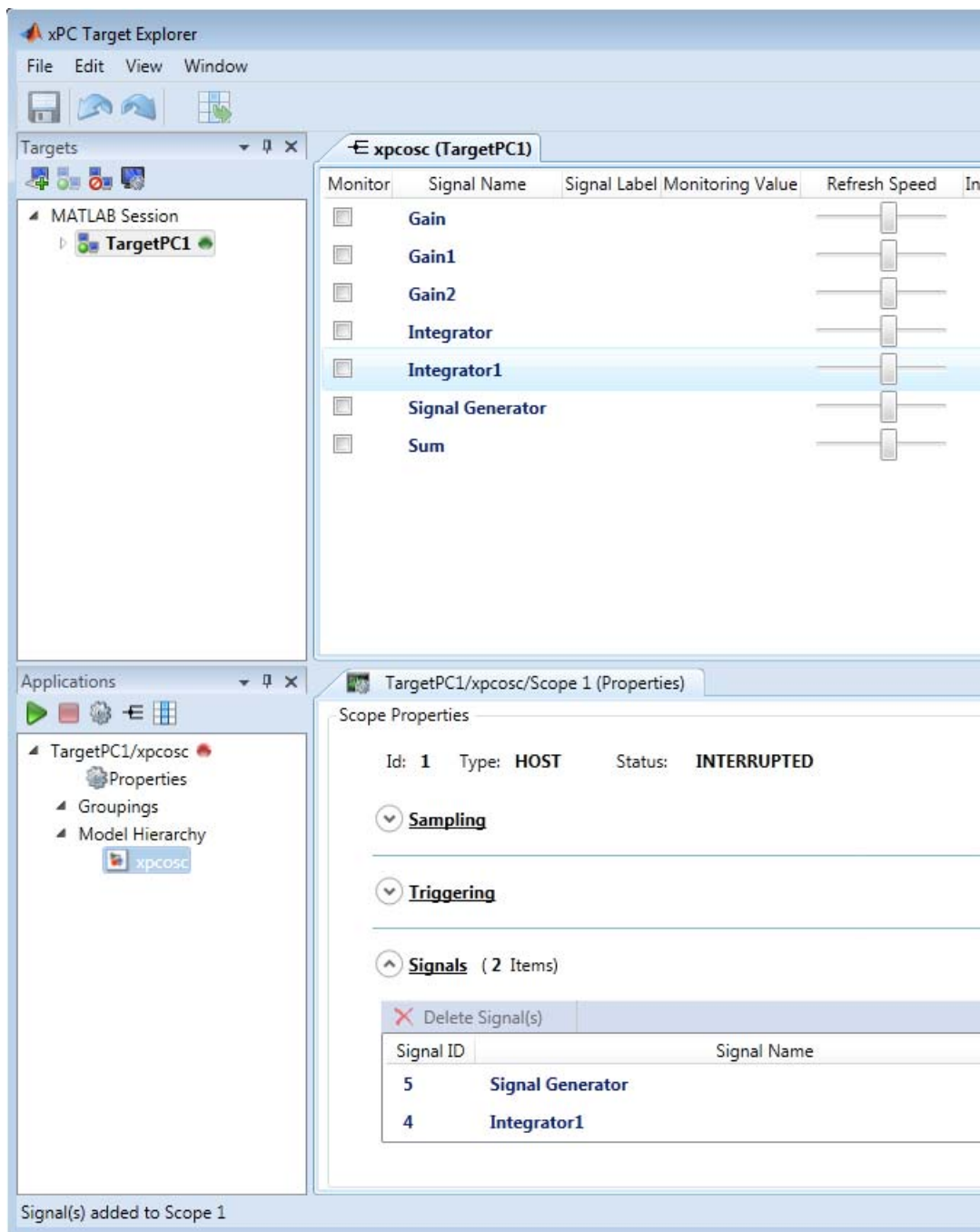
10 To view the host scope, select **Scope 1** and then click the View Scope icon ![icon] on the toolbar.

The Host Scope Viewer window opens as a separate tab. The signals you add to the scope appear at the top right of the viewer.

11 To start execution, click the target application and then click the Start icon ![icon] on the toolbar.

The application starts running. No output appears on the host scope viewer.

12 To start **Scope 1**, click **Scope 1** in the **Scopes** pane and then click the Start Scope icon ![icon] on the toolbar.

Output for signals `Signal Generator` and `Integrator1` appears on the host scope viewer.
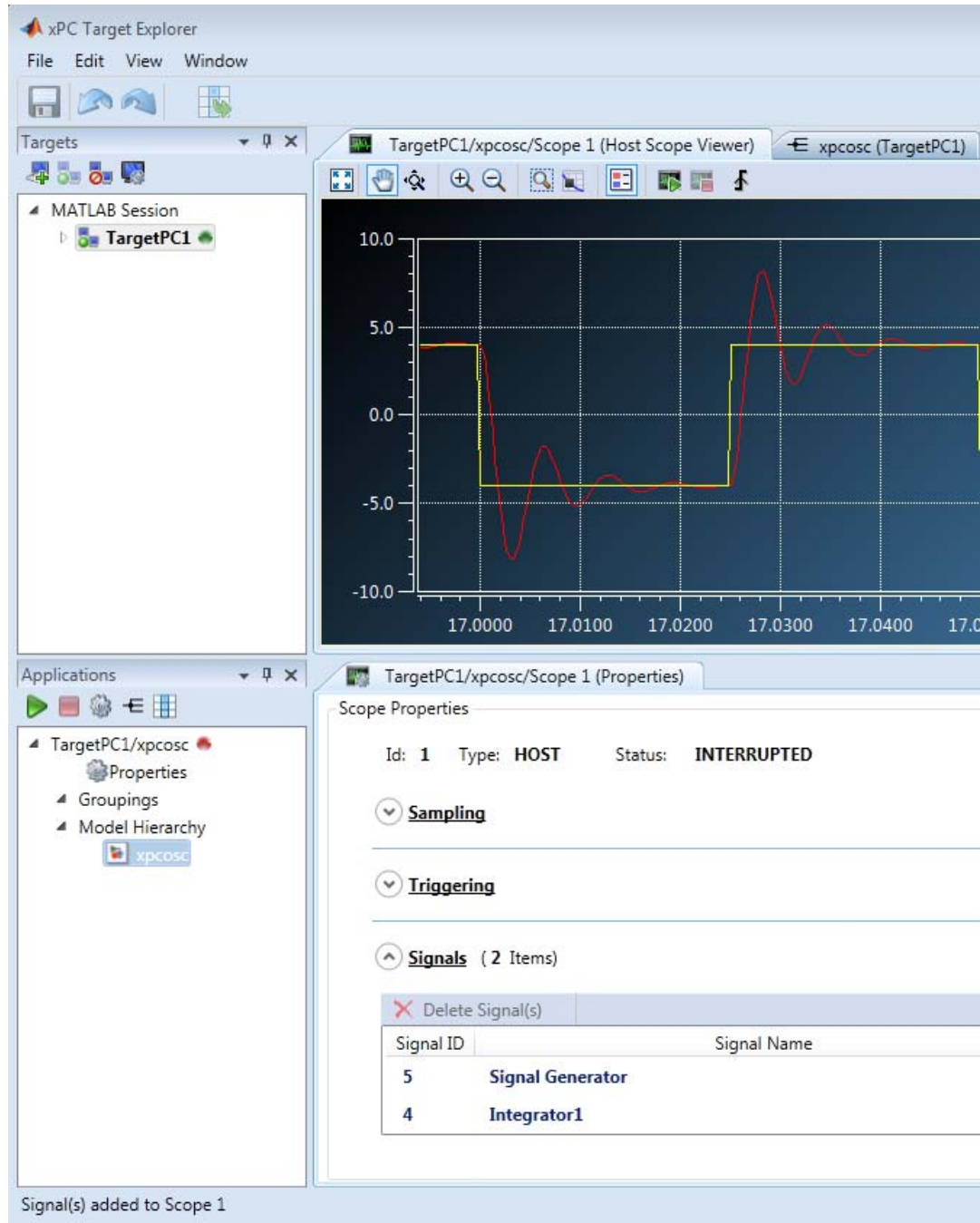
13 To stop **Scope 1**, click **Scope 1** in the **Scopes** pane and then click the Stop Scope icon ![icon] on the toolbar.

The signals shown on the target computer stop updating while the target application continues running. The target computer monitor displays a message like this one:

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

14 To stop execution, click the target application and then click the Stop icon ![icon] on the toolbar.

The target application on the target computer stops running, and the target computer displays messages like this message:

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

- To configure the host scope viewer, see "Configure the Host Scope Viewer" on page 5-65

- You can create a virtual host scope from the scope types list by clicking **Add Scope** next to scope type **Host Scopes**.

- To group signals, see "Create Signal Groups Using xPC Target Explorer" on page 5-54.

- To configure data sampling, see "Configure Scope Sampling Using xPC Target Explorer" on page 5-37.

- To configure interactive scope triggering, see "Trigger Scopes Interactively Using xPC Target Explorer" on page 5-40.

- To configure noninteractive scope triggering, see "Trigger Scopes Noninteractively Using xPC Target Explorer" on page 5-44.

- To make the active xPC Target Explorer workspaces visible at the same time, click and hold the tab for one workspace and drag it down until the following icon appears in the middle of the dialog box: ![icon] . Continue to drag until the cursor reaches the required quadrant, and then release the mouse button. Repeat for the other workspaces.

- If a block is unnamed, xPC Target Explorer does not display signals or a node for that block. To reference such a block, provide an alphanumeric

name for that block, rebuild and download the model to the target computer, and then reconnect the MATLAB session to the target computer.

Configure the Host Scope Viewer

You can customize the viewer for each host scope to facilitate your interaction with the running model.

This procedure uses the model xpcosc as an example. You must have already completed the procedure in "Create Host Scopes Using xPC Target Explorer" on page 5-58. Target execution and scopes must be stopped.

1 Start execution (on the **Applications** toolbar).

2 To start **Scope 1**, click the Start icon on the Host Scope Viewer toolbar.

3 To trigger **Scope 1**, click the Trigger icon on the Host Scope Viewer toolbar.

   To interactively trigger a capture using the Trigger icon , you must set the scope **Trigger Mode** to Software or Scope. See "Trigger Scopes Interactively Using xPC Target Explorer" on page 5-40.

4 In the xPC Target Host Scope Viewer, right-click anywhere in the axis area of the viewer and then click **Edit**.

   The Host Scope Viewer display parameter icons become enabled on the toolbar.

5 Adjust the Host Scope Viewer display using:

   • Auto Scale — To scale the display to accommodate the top and bottom of the Y-axis.

   • Axes Scroll — To move the content up and down and right and left relative to the axes. The axes scroll as required.

   • Axes Zoom — To stretch and compress the X-axis and Y-axis.

   • Zoom In — To zoom in on the current center of the display.

   • Zoom Out — To zoom out from the current center of the display.

   • Zoom Box — To select an area of interest in the display. When you release the mouse button, the display zooms in upon the selected area.

- Data Cursor — To display data values using a set of cross-hairs in the display.

  Data is displayed as the pair `x-value,y-value`, indicating the value at that point on the display. You can drag the center of the cross hairs and observe the value at each point.

- Legends — To toggle display of the signal names.

6 To stop **Scope 1**, click the Stop icon on the Host Scope Viewer toolbar.

7 Stop execution (on the **Applications** toolbar).

Configure Target Scopes Using MATLAB Language

Creating a scope object allows you to select and view signals using xPC Target functions instead of the xPC Target graphical user interface.

This procedure uses the Simulink model xpcosc as an example. To do this procedure, you must have already built the target application forxpcosc and assigned tg to the target computer. It describes how to trace signals with target scopes.

1 Start running your target application. Type:

```
tg.start
```

The target computer displays the following message:

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

2 To get a list of signals, type:

```
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 are:

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

For more information, see "Monitor Signals Using MATLAB Language" on page 5-8.

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

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

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. The scope properties Time and Data are not accessible with a target scope.

```
xPC Scope Object
   Application         = xpcosc
   ScopeId             = 1
   Status              = Interrupted
   Type                = Target
   NumSamples          = 250
   NumPrePostSamples   = 0
   Decimation          = 1
   TriggerMode         = FreeRun
   TriggerSignal       = -1
   TriggerLevel        = 0.000000
   TriggerSlope        = Either
   TriggerScope        = 1
   TriggerSample       = -1
   Mode                = Redraw (Graphical)
   YLimit              = Auto
   Grid                = On
   Signals             = no Signals defined
```

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

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

The target computer displays the following messages:

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

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

6 Start the scope. For example, to start the scope sc1, type:

```
sc1.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.

7 Stop the scope. Type:

```
sc1.stop
```

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

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

8 Stop the target application. In the MATLAB window, type:

```
tg.stop
```

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

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

Trace Signals Using Simulink External Mode

You can use Simulink external mode to establish a communication channel between your Simulink block diagram and your target application. The block diagram becomes a graphical user interface to your target application. Simulink scopes can display signal data from the target application, including from models referenced inside a top model. You can control which signals to upload through the External Signal & Triggering dialog box (see "Signal Selection" and "Control External Mode Operations").

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

This procedure uses the model xpcosc as an example. xpcosc contains a Simulink Scope block.

1 In the MATLAB window, type xpcosc.

2 In the Simulink window, from the **Code** menu, select **External Mode Control Panel**.

3 In the External Mode Control Panel dialog box, click the **Signal & Triggering** button.

4 In the External Signal & Triggering dialog box, set the **Source** parameter to manual.

5 Set the **Mode** parameter to normal. In this mode, the scope acquires data continuously.

6 Select the **Arm when connecting to target** check box.

7 In the **Delay** box, enter 0.

8 In the **Duration** box, enter the number of samples for which external mode is to log data, for example 1000.

The External Signal & Triggering dialog box looks like this figure.
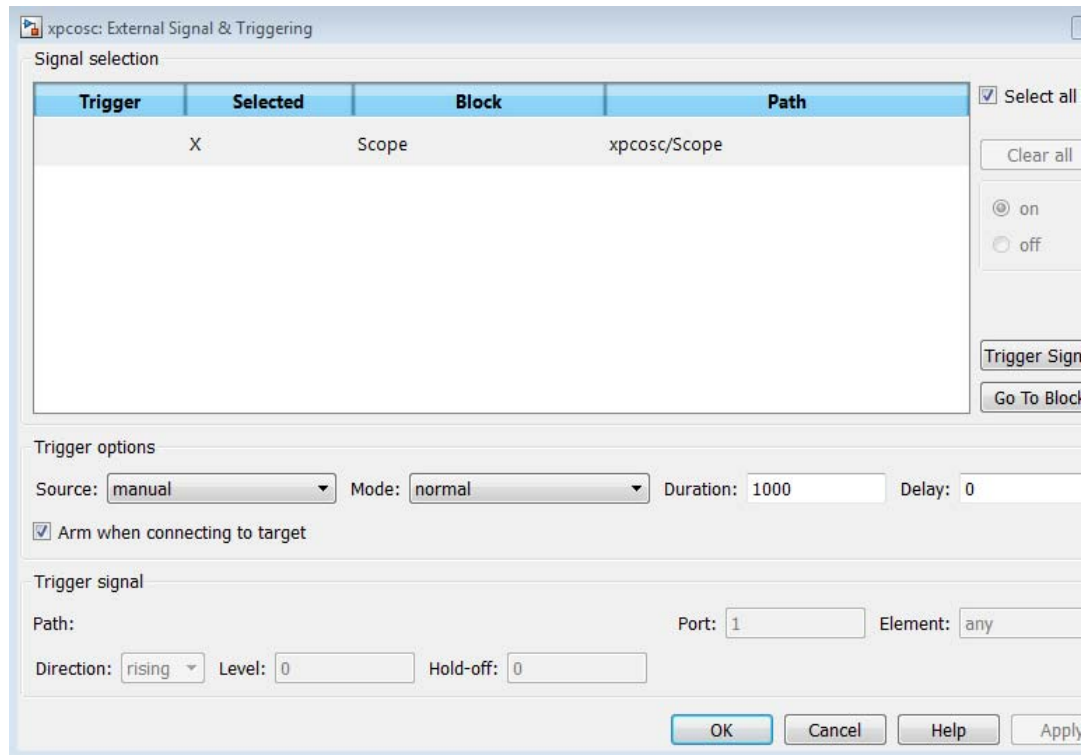
9 Click **Apply**, and then **Close**.

10 In the External Mode Control Panel dialog box, click **OK**.

11 In the Simulink toolbar, increase the simulation stop time to, for example,
   50.

12 From the **File** menu, select **Save As** and enter a file name. For example,
   enter my_xpc_osc6, and then click **OK**.

13 To build and download the target application, click the Build icon 🔢 on
   the Simulink toolbar.

   The xPC Target software downloads the target application to the default
   target computer.

5-71

14 In the Simulink window, click **Simulation > Mode > External**. A check mark appears next to the menu item **External**, indicating that Simulink external mode is activated.

15 If a Scope window is not displayed for the Scope block, double-click the Scope block.

16 In the **Scope** window, click the Connect To Target icon on the toolbar.

   The current Simulink model parameters are downloaded from the host computer to the target application.

17 To start the simulation, click the Run icon on the toolbar.

   The target application begins running on the target computer. The Scope window displays plotted data.
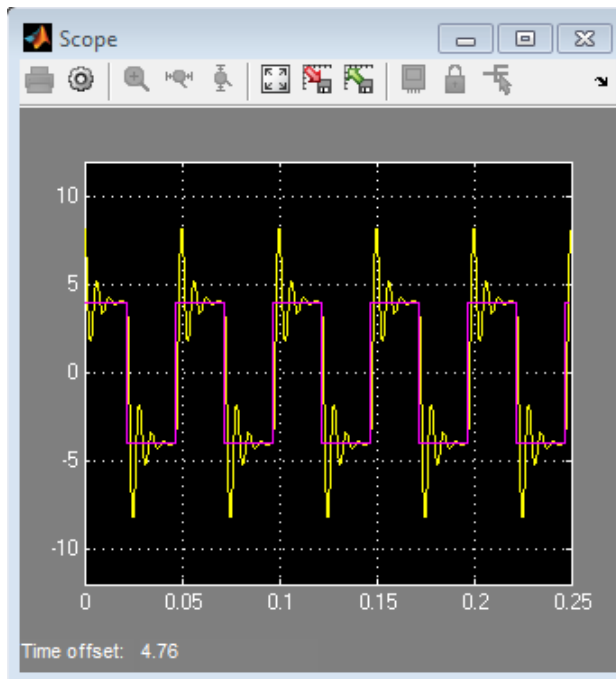
18 To stop the simulation, click the Stop icon ⬛ on the toolbar.

External Mode Usage

- When setting up signal triggering (Source set to signal), you must explicitly specify the element number of the signal in the **Trigger signal:Element** box. If the signal is a scalar, enter a value of 1. If the signal is a wide signal, enter a value from 1 to 10. When uploading xPC Target signals to Simulink scopes, do not enter Last or Any in this box.

- The **Direction:Holdoff** value does not affect the xPC Target signal uploading feature.

- Attempting to upload information from buses and virtual signals inside a reference model generates a warning.

Trace Signals Using a Web Browser

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

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

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 target scope is created and displayed on the target computer. The **Scopes** pane displays a list of the scopes present. You can add a new scope, remove existing scopes, and configure the scopes from this page.

To create a host scope, use the drop-down list next to the **Add Scope** button to select Host. This item is set to Target by default.

3 Click the **Edit** button.

From the scope editing pane, you can configure 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 **Apply**. A **Remove Existing Signals** list is added above the **Add New Signals** list.

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

When a host scope is stopped (**Scope State** is set to Interrupted) or finishes one cycle of acquisition (**Scope State** is set to Finished), the **Get Data** button becomes available. If you click this button, the scope data is retrieved in comma-separated value (CSV) format. The signals in the scope are spread across columns. Each row corresponds to one sample of acquisition. The first column corresponds to the time each sample was acquired.

If **Scope State** is set to `Interrupted`, the scope was stopped before it completed a full cycle of acquisition. The number of rows in the CSV data still correspond to a full cycle. The last few rows (for which data was not acquired) are set to `0`.

Signal Logging Basics

Signal logging acquires signal data during a real-time run and stores it on the target computer. After you stop the target application, you transfer the data from target computer to host computer for analysis. Signal logging is also known as real-time data streaming to the target computer. You can plot and analyze the data, and later save it to a disk on the host computer.

xPC Target signal logging samples at the base sample time. If you have a model with multiple sample rates, add xPC Target scopes to the model to sample signals at the required sample rates.

- The xPC Target software does not support logging data with decimation.
- xPC Target Explorer works with multidimensional signals in column-major format.
- Some signals are not observable. See "Nonobservable Signals and Parameters" on page 5-140.

You can log signals using file scopes in the model, virtual file scopes in xPC Target Explorer, outports in the model, MATLAB language, and a web browser.

Configure File Scope (xPC) Blocks

xPC Target includes a specialized Scope (xPC) block that you can configure to save signal and time data to a file on the target computer hard drive, flash drive, or removable drive. Add a Scope (xPC) block to the model, select **Scope type** File, and then configure the other parameters as described in the following procedure.

- Do not confuse xPC Target Scope blocks with standard Simulink Scope blocks.

- For more information about using xPC Target Scope blocks, see "xPC Target Scope Usage" on page 5-25.

- For more information about target scopes, see "File Scope Usage" on page 5-83.

- This procedure uses the model my_xpc_osc2 as an example. To access the example folder, type:

  ```
  addpath (fullfile(matlabroot, 'help', 'toolbox', 'xpc',
       'examples'));
  ```

1 In the MATLAB window, type my_xpc_osc2.

2 In the Simulink block diagram, double-click the block labeled Scope (xPC).

  The Block Parameters: Scope (xPC) dialog box opens. By default, the target scope dialog box is displayed.

3 In the **Scope number** box, a unique number is displayed that identifies the scope. This number is incremented each time you add a new xPC Target scope.

  This number identifies the xPC Target Scope block and the scope screen on the host or target computer.

4 From the **Scope type** list, select File. The updated dialog box opens.

5 When the target application is downloaded and started, select the **Start scope when application starts** check box to start a scope. The scope window opens automatically.

In Stand Alone mode, this setting is mandatory because the host computer is not available to issue a command to start scopes.

6 In the **Number of samples** box, enter the number of values to be acquired in a data package. This parameter works in conjunction with the **AutoRestart** check box. If you select the **AutoRestart** box, the file scope collects data up to Number of samples, and then starts over again, overwriting the buffer. If you do not select the **AutoRestart** box, the file scope collects data only up to Number of samples, and then stops.

7 In the **Number of pre/post samples** box, enter the number of samples to save or skip. To save N samples before a trigger event, specify the value N. To skip N samples after a trigger event, specify the value N. The default is 0.

8 In the **Decimation** box, enter a value to indicate that data is collected at each sample time (1) or at less than every sample time (2 or greater).

9 From the **Trigger mode** list, select FreeRun, Software Triggering, Signal Triggering, or Scope Triggering.

- If you select FreeRun or Software Triggering, you do not need to enter additional parameters.

- If you select Signal Triggering, then enter the following additional parameters, as required:

  - In the **Trigger signal** box, enter the index of a signal previously added to the scope.

    This parameter does not apply if the **Add signal port to connect a signal trigger source** check box is selected.

  - (Alternatively) Click the **Add signal port to connect a signal trigger source** check box, then connect an arbitrary trigger signal to the port Trigger signal.

  - In the **Trigger level** box, enter a value for the signal to cross before triggering.

  - From the **Trigger slope** list, select one of Either, Rising, or Falling.

- If you select Scope Triggering, then enter the following additional parameters, as required:

–   In the **Trigger scope number** box, enter the scope number of a Scope block. If you use this trigger mode, you must also add a second Scope block to your Simulink model.

–   If you want the scope to trigger on a specific sample of the other scope, enter a value in the text box **Sample to trigger on (-1 for end of acquisition)**. The default value is 0, and indicates that the triggering scope and the triggered (current) scope start simultaneously.

    For more information on this field, see "Trigger One Scope with Another Scope" on page 7-21.

10 In the **Filename** box, enter a name for the file to contain the signal data.

By default, the target computer writes the signal data to C:\data.dat. For more about files and file names, see "File Scope Usage" on page 5-83.

11 From the **Mode** list, select either Lazy or Commit.

With the Commit mode, each file write operation simultaneously updates the FAT entry for the file. The file system maintains the actual file size after each write. With the Lazy mode, the FAT entry is updated only when the file is closed.

If your system stops responding, you lose WriteSize bytes of data.

12 In the **WriteSize** box, enter the block size, in bytes, of the data chunks. This parameter specifies that a memory buffer of length Number of samples is written to the file in chunks of size WriteSize. By default, this parameter is 512 bytes. Using a block size that is the same as the disk sector size improves performance.

If your system stops responding, you lose WriteSize bytes of data.

13 In the **Number of samples** box, enter the number of values to be acquired in a data package.

14 Select the **AutoRestart** check box to enable the file scope to collect up to Number of samples data samples, write the buffer to the signal data file, and then start over again, appending the new data to the end of the signal data file. Clear the **AutoRestart** check box to have the file scope collect

up to Number of samples data samples, write the buffer to the signal data file, and then stop.

If the named signal data file already exists, the xPC Target software overwrites the old data with the new signal data.

The file scope dialog box looks like this figure.

Sink Block Parameters: Scope (xPC)

xpcscopeblock (mask) (link)

xPC Target Scope
Configure scope to acquire signal data.
Scope can be of type target, host, or file.

Parameters

Scope number:

1

Scope type: File

☑ Start scope when application starts

Number of samples:

1000

Number of pre/post samples:

0

Decimation:

1

Trigger mode: FreeRun

Filename:

data.dat

Mode: Lazy

WriteSize:

512

☐ AutoRestart

OK   Cancel   Help   Apply

15 Click **OK**.

16 From the **File** menu, click **Save As**. The model is saved as `my_xpc_osc2`.

File Scope Usage

- xPC Target supports eight file scopes. Each file scope can contain as many signals as the target computer has resources to support them.

- With file scopes, after you run the target application, the xPC Target software generates a signal data file on the target computer, even if it is running in Stand Alone mode. To access the contents of the signal data file that a file scope creates, use the xPC Target file system object (xpctarget.fs) from a host computer MATLAB window. To view or examine the signal data, 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 "Using xpctarget.fs Objects" on page 8-10. Saving signal data to files lets you recover signal data from a previous run in the event of system failure.

  The signal data file can quickly increase in size. 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.

- File names on the target computer are limited to 8 characters in length, not counting the file extension. If the name is longer than 8 characters, the software truncates it to 6 characters and adds '~1' to the end of the file name.

  If you enter just the file name, the file appears in folder C:\. To put the file in a folder, you must create the folder separately using the target computer command line or MATLAB language (see xpctarget.fsbase.mkdir).

  To configure the scope to generate multiple, dynamically named files in one session, see "Log Signal Data into Multiple Files" on page 5-93.

- Both the Lazy and Commit settings of the **Mode** box cause the model to open a file, write signal data to the file, and then close that file at the end of the session. With the Commit mode, each file write operation simultaneously updates the FAT entry for the file. This mode is slower, but the file system maintains the actual file size after each write. With the 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 stops responding before the file is closed, the file system might not know the actual file size (the file contents, however, will be intact). If the system stops responding, you lose an amount of data equivalent to the setting of the **WriteSize** parameter.

- For a file scope, the scope acquires data and writes it to the file named in the **FileName** parameter. The scope acquires the first N samples into a memory buffer of size given by the **Number of Samples** parameter. The scope writes data from the memory buffer to the file in blocks of size given by the **WriteSize** parameter.

  If you select the **AutoRestart** check box, the scope starts over, overwriting the memory buffer. The additional data is appended to the end of the existing file.

  If you do not select the **AutoRestart** box, the scope collects data only up to the number of samples, and then stops.

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

  The number of samples N to log after triggering an event is equal to the value that you entered in the **Number of Samples** parameter. If you stop and then start the scope again, the data in the file is overwritten with the new data.

Create File Scopes Using xPC Target Explorer

You can create a virtual file scope on the target computer using xPC Target Explorer. These scopes have the full capabilities of the Scope (xPC) block in File mode, but do not persist past the current execution.

Note For information on using file scope blocks, see "Configure File Scope (xPC) Blocks" on page 5-78 and "File Scope Usage" on page 5-83.

This procedure uses the model xpcosc as an example. You must have already completed the following setup:

1 Built and downloaded the target application to the target computer using

Simulink (on the toolbar).

2 Run xPC Target Explorer (command xpcexplr).

3 Connected to the target computer in the **Targets** pane (on the toolbar).

4 Set property **Stop time** to inf in the **Applications** pane (on the toolbar).

To configure a virtual file scope:

1 In the **Scopes** pane, expand the **xpcosc** node.

2 To add a file scope, select **File Scopes**, and then click the Add Scope icon on the toolbar.

3 Expand **Scope 1**, and then click the Properties icon on the toolbar.

4 In the **Scope Properties** pane, click **Signals**.

Add signals from the **Target Applications** Signals workspace.

5 In the **Target Applications** pane, expand both the target application node and the node **Model Hierarchy**.

6 Select the model node and then click the View Signals icon ∈ on the toolbar.

7 In the Signals workspace, to add signal `Signal Generator` to **Scope1**, click the down arrow next to the Scopes icon 🔽 in its Actions column. A list of scope types appears. **Scope 1** appears under node **File Scopes**.

8 Click the Add Signal(s) icon ➕ next to **Scope1** under the **File Scopes** node.

9 Add signal `Integrator1` to **Scope 1** in the same way.

10 In the **Scope Properties** pane, click **File**.

11 Enter a name in the **File name** text box, for example `scope1.dat`.

12 To start execution, click the target application and then click the Start icon ▶ on the toolbar.

13 To start **Scope 1**, click **Scope 1** in the **Scopes** pane, and then click the Start Scope icon 🔼 on the toolbar.

14 To stop **Scope 1**, click **Scope 1** in the **Scopes** pane, and then click the Stop Scope icon 🔽 on the toolbar.

For file scopes, before adding or removing signals, you must stop the scope first.

15 To stop execution, click the target application, and then click the Stop icon 🔴 on the toolbar.

16 To view the file that you generated, in the **Targets** pane, expand the target computer and then double-click **File System**.

17 Select `C:\`. The dialog box looks like this figure.

- To retrieve the file from the target computer for analysis, see "Using xpctarget.fs Objects" on page 8-10

- To rename file SCOPE1.DAT, right click the file name, select **Rename**, type the new name in the text box, and then click **Enter**.

  To delete file SCOPE1.DAT, right click the file name and select **Delete**.

- You can create a virtual file scope from the list of scope types by clicking **Add Scope** next to scope type **File Scopes**.

- To group signals, see "Create Signal Groups Using xPC Target Explorer" on page 5-54.

- To configure data sampling, see "Configure Scope Sampling Using xPC Target Explorer" on page 5-37.

- To configure scope triggering, see "Trigger Scopes Interactively Using xPC Target Explorer" on page 5-40 and "Trigger Scopes Noninteractively Using xPC Target Explorer" on page 5-44.

- To make the active xPC Target Explorer workspaces visible at the same time, click and hold the tab for one workspace and drag it down until the

  following icon appears in the middle of the dialog box: . Continue to drag until the cursor reaches the required quadrant, and then release the mouse button. Repeat for the other workspaces.

- If a block is unnamed, xPC Target Explorer does not display signals or a node for that block. To reference such a block, provide an alphanumeric name for that block, rebuild and download the model to the target computer, and reconnect the MATLAB session to the target computer.

Configure File Scopes Using xPC Target Explorer

You can configure your file scopes to facilitate data logging. You can configure a file scope whether you added a Scope (xPC) block to your model or added the scope at run time.

This procedure uses the model xpcosc as an example. You must have already completed the procedure in "Create File Scopes Using xPC Target Explorer" on page 5-85. Target execution and scopes must be stopped.

1 Select **Scope 1**, and then open the Properties pane (![icon] on the **Scopes** toolbar).

2 In the **Scope 1** Properties pane, click **File**.

3 Enter a name in the **File name** text box, for example scope2.dat.

File names on the target computer are limited to 8 characters in length, not counting the file extension. If the name is longer than 8 characters, the software truncates it to 6 characters and adds '~1' to the end of the file name.

If you enter just the file name, the file appears in folder C:\. To put the file in a folder, you must create the folder separately using the target computer command line or MATLAB language (see xpctarget.fsbase.mkdir).

If a file with this name already exists when you start the file scope, the file scope overwrites the old data with the new data.

4 Select **File mode** Commit.

The default **File mode** is Lazy. In both Lazy and Commit mode, the kernel opens a file, writes signal data to the file, and closes that file at the end of the session.

- In Commit mode, each file write operation simultaneously updates the FAT entry for the file. This mode is slower than Lazy mode, but the file system maintains the actual file size after each write.

- In Lazy mode, the FAT entry is updated only when the file is closed and not during each file write operation. This mode is faster than Commit mode, but if the system stops responding before the file is closed, the file

system might not know the actual file size, even though the contents will be intact. If the system stops responding, you lose an amount of data equivalent to the setting of the **Write Size** parameter.

5 Select the **AutoRestart** check box.

- When you select **AutoRestart**, the file scope collects data up to the number of samples, and then restarts. It appends the new data to the end of the file.

- When you clear **AutoRestart**, the file scope collects data up to the number of samples, and then stops.

6 Leave the **Dynamic File Mode** check box cleared.

For information on using **Dynamic File Mode** to generate multiple, dynamically named files in one session, see "Log Signal Data into Multiple Files" on page 5-93.

7 Leave **Write Size** set to the default value of 512.

Using a block size that is the same as the disk sector size improves performance.

8 Leave **Max write file size** set to the default value, which is a multiple of **Write Size**.

9 Start execution (on the **Applications** toolbar).

10 Start **Scope 1**(on the **Scopes** toolbar). Let it run for up to a minute.

11 Stop **Scope 1** (on the **Scopes** toolbar).

12 Stop execution (on the **Applications** toolbar).
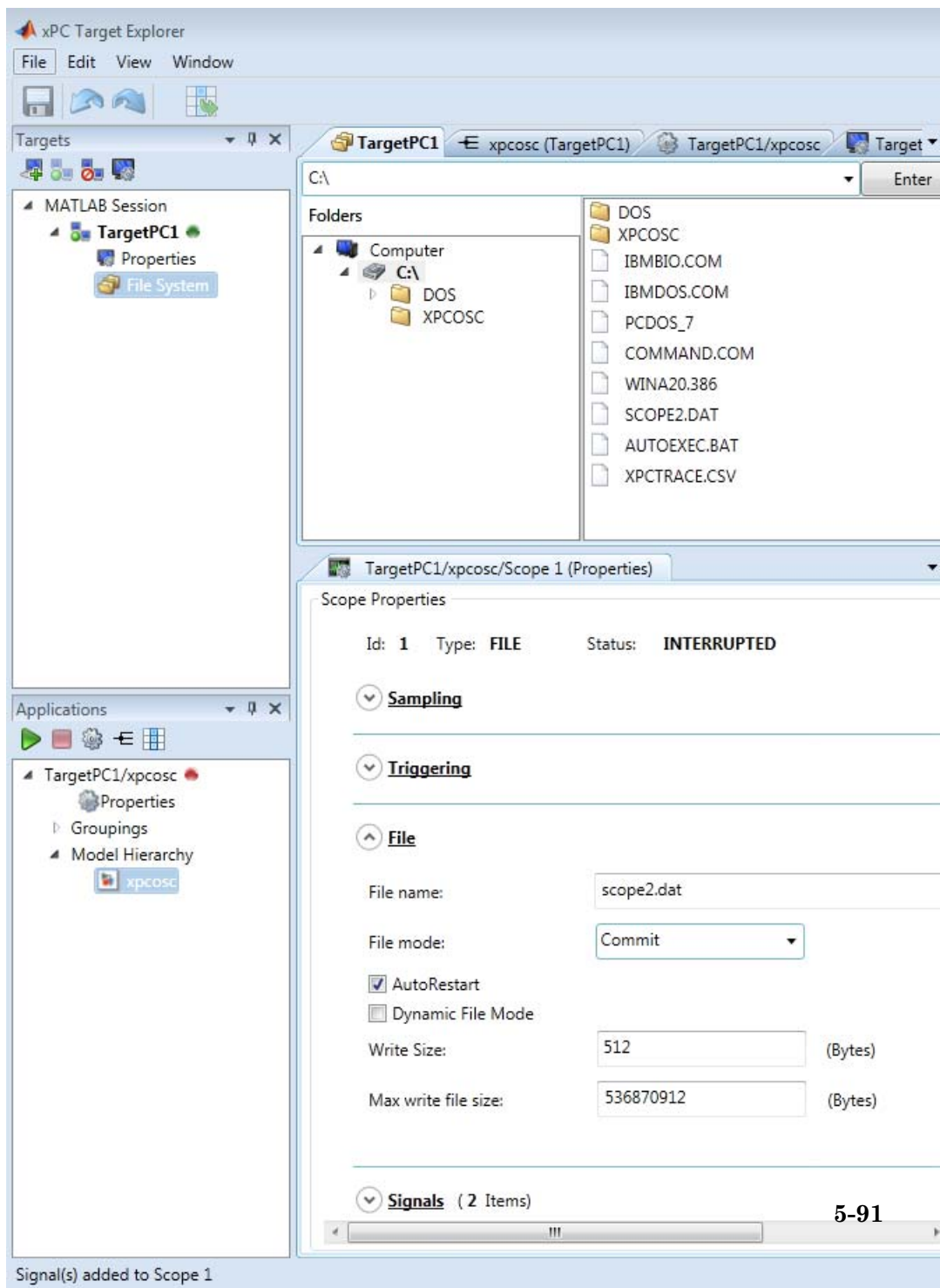
- To rename file SCOPE2.DAT, right-click the file name, select **Rename**, type the new name in the text box, and then click **Enter**.

- To delete file SCOPE2.DAT, right-click the file name and select **Delete**.

- For information about how to retrieve the file from the target computer for analysis, see "Using xpctarget.fs Objects" on page 8-10

Log Signal Data into Multiple Files

You can acquire signal data to store in multiple, dynamically named files on the target computer. You can then examine one file while the scope continues to acquire data to store in other files. To acquire data for multiple files, add a file scope to the target application, and then configure that scope to log signal data to multiple files.

You must use model xpcosc and have completed the setup tasks in "Create File Scopes Using xPC Target Explorer" on page 5-85.

1 In xPC Target Explorer, in the **Scopes** pane, expand the **xpcosc** node.

2 Select **File Scopes** and expand node **File Scopes**.

3 Expand **Scope 1** and then click the Properties icon ![icon] on the toolbar.

4 In the **Scope Properties** pane, click **File**.

5 To enable the file scope to create multiple log files based on the same name, in the **File name** box, enter a name like scope1_<%>.dat.

This sequence directs the software to create up to nine log files, scope1_1.dat to scope1_9.dat, on the target computer file system.

You can configure the file scope to create up to 99999999 files (<%%%%%%%>.dat). The length of a file name, including the specifier, cannot exceed eight characters. See the file name description in xpctarget.xpc.get (target application object).

6 Select the **AutoRestart** and **Dynamic File Mode** check boxes.

7 In the **Max write file size** box, enter a value to limit the size of the signal log files. This value must be a multiple of the **Write Size** value. For example, if the write size is 512, enter 4096 to limit each log file size to 4096 bytes.

8 To start execution, click the target application and then click the Start icon ![icon] on the toolbar.

9 To start **Scope 1**, click **Scope 1** in the **Scopes** pane and then click the Start Scope icon on the toolbar.

Let **Scope 1** run for up to a minute.

10 To stop **Scope 1**, click **Scope 1** in the **Scopes** pane and then click the Stop Scope icon on the toolbar.

11 To stop execution, click the target application and then click the Stop icon on the toolbar.

12 To view the files that you generated, in the **Targets** pane, expand the target computer, and then double-click **File System**.

13 Select C:\. The dialog box looks like this figure.

of **Max write file size**), the software closes SCOPE1_1.DAT and creates
SCOPE1_2.DAT, SCOPE1_3.DAT, and so on until it fills the last log file,
SCOPE1_9.DAT. If the target application continues to collect data after
the software closes SCOPE1_9.DAT, the software reopens SCOPE1_1.DAT,
SCOPE1_2.DAT, and so on, overwriting the existing contents.

For information about how to retrieve the file from the target computer for
analysis, see "Using xpctarget.fs Objects" on page 8-10

Configure Outport Logging Using xPC Target Explorer

To use xPC Target Explorer for signal logging, add an Outport block to your Simulink model. Activate logging on the **Data Import/Export** pane in the Configuration Parameters dialog box.

This procedure begins with tutorial model xpc_osc3:

1 In the MATLAB window, type xpc_osc3. The xpc_osc3 model opens.

2 In the Simulink window, select and delete the xPC Target Scope block and its connecting signal.

3 Click **Simulation > Model Configuration Parameters**.

4 Select node **Data Import/Export**.

5 Select the **Signal logging** check box.

6 In **Signal logging format**, select value Dataset.
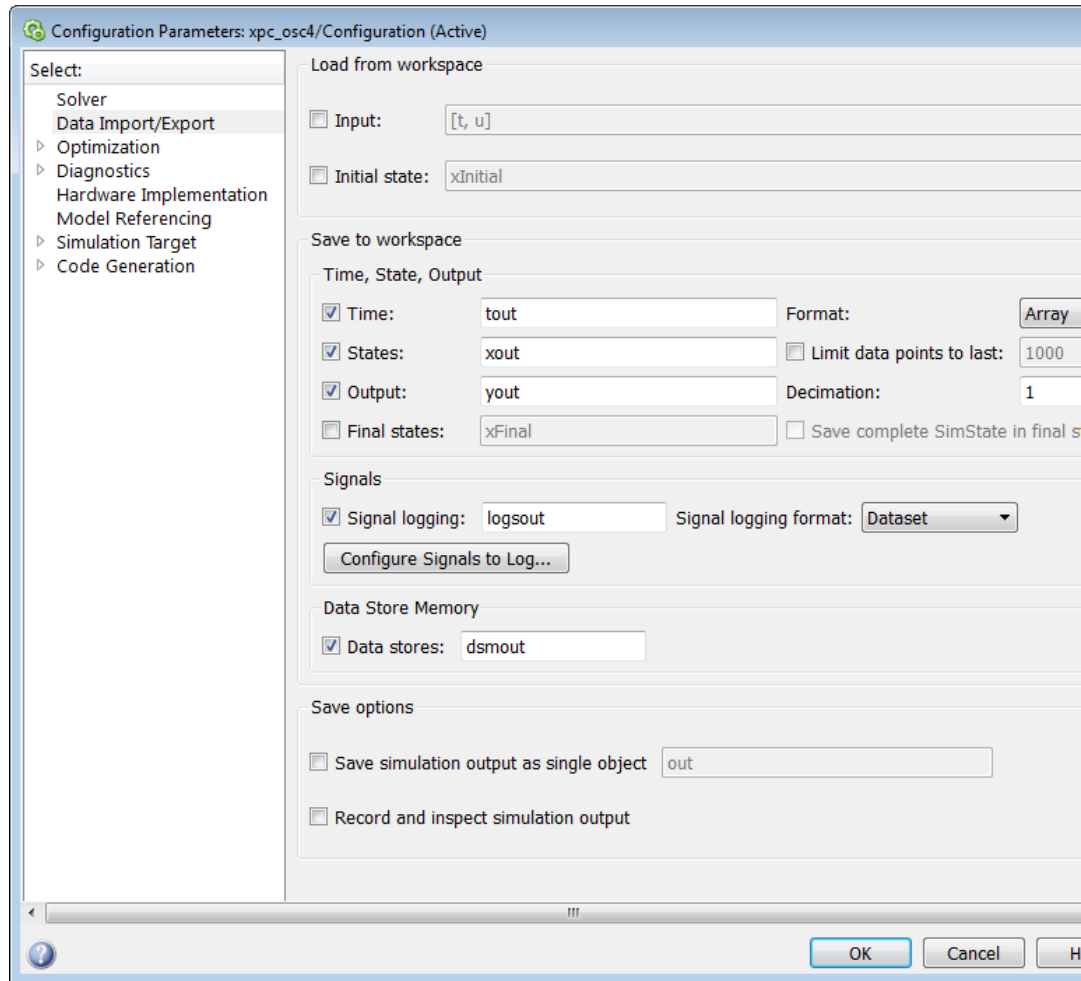
7 From the **File** menu, click **Save as**. Enter xpc_osc4, and then click **Save**.

8 Click **OK**.

9 In the Simulink window, click the Build Model icon on the toolbar.

10 Run xPC Target Explorer using command xpcexplr.

11 Connect to the target computer in the **Targets** pane using the Connect icon ▦ on the toolbar.

12 To start execution, click the target application, and then click the Start icon ▶ on the toolbar.

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

13 To stop execution, click the target application, and then click the Stop icon ■ on the toolbar.

14 Plot the signals from the Outport block and the states. In the MATLAB window, type:

```
plot(tg.TimeLog,tg.Outputlog)
```

Values for the logs are uploaded to the host computer from the target application on the target computer. To upload part of the logs, see the target object method xpctarget.xpc.getlog.

The plotted output looks like this figure.
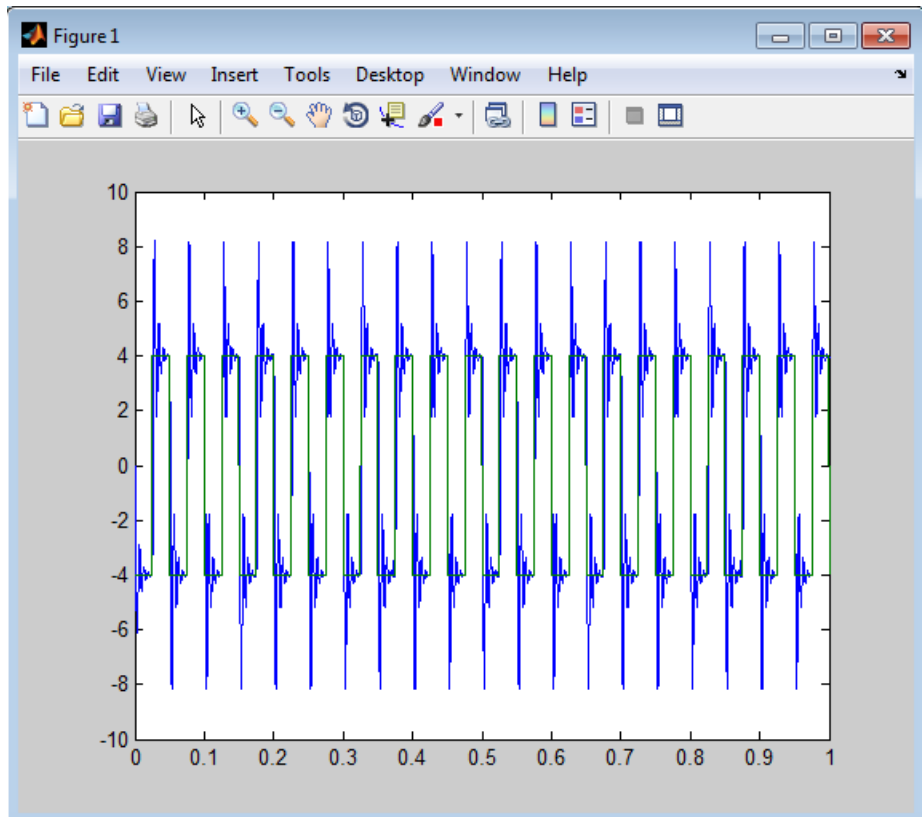
Configure Outport Logging Using MATLAB Language

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. Before you configure data logging, you must complete the following setup:

1 Before you build the target application, add Outport blocks to your Simulink model. In the **Data Import/Export** pane of the Configuration Parameters dialog box, select the **Save to workspace** check box . See "Configure Simulation Parameters".

2 To plot the task execution time, in the **xPC Target options** pane of the Configuration Parameters dialog box, verify that the **Log Task Execution Time** check box is selected. This check box is selected by default. See "Add xPC Target Scope Block".

3 In the **xPC Target options** pane of the Configuration Parameters dialog box, set **Signal logging buffer size in doubles** to a value large enough to accommodate the logged signals. The default is `100000`. If the default buffer size is not large enough, approximate the size using this formula:

```
Buffer size in doubles = 90% * Memory / sizeof(double)
```

`Memory` is the number of bytes available on the target computer after the kernel starts. It is displayed in the upper-left corner of the target computer screen. For example, for a `Memory` value of `2044MB`, set **Signal logging buffer size in doubles** to `255500000`.

The xPC Target software calculates the number of samples `N` for a signal as the value of **Signal logging buffer size in doubles** divided by the number of logged signals (1 time, 1 task execution time ([TET]), number of outputs, number of states). The scopes copy the last `N` samples from the log buffer to the target object logs (`tg.TimeLog`, `tg.OutputLog`, `tg.State`Log, and `tg.TETLog`).

After you build, download, and run a target application, you can plot the state and output signals. This procedure uses the Simulink model `xpc_osc4` as an example. You must have already built and downloaded the target application for that model.

1 Assign `tg` to the target computer. In the MATLAB window, type:

```
tg=xpc
```

2 Start the target application. In the MATLAB window, type:

```
tg.start
```

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` has one Outport block, labeled 1. There are two states. This Outport block shows the signals leaving the blocks labeled Integrator1 and Signal Generator.

3 Plot the signals from the Outport block and the states. In the MATLAB window, type:

```
plot(tg.TimeLog,tg.Outputlog)
```

Values for the logs are uploaded to the host computer from the target application on the target computer. To upload part of the logs, see the target object method `xpctarget.xpc.getlog`.

The plot shown is the result of a real-time execution. To compare this plot with a plot for a non-real-time simulation, see "Simulate Simulink Model Using MATLAB Language".
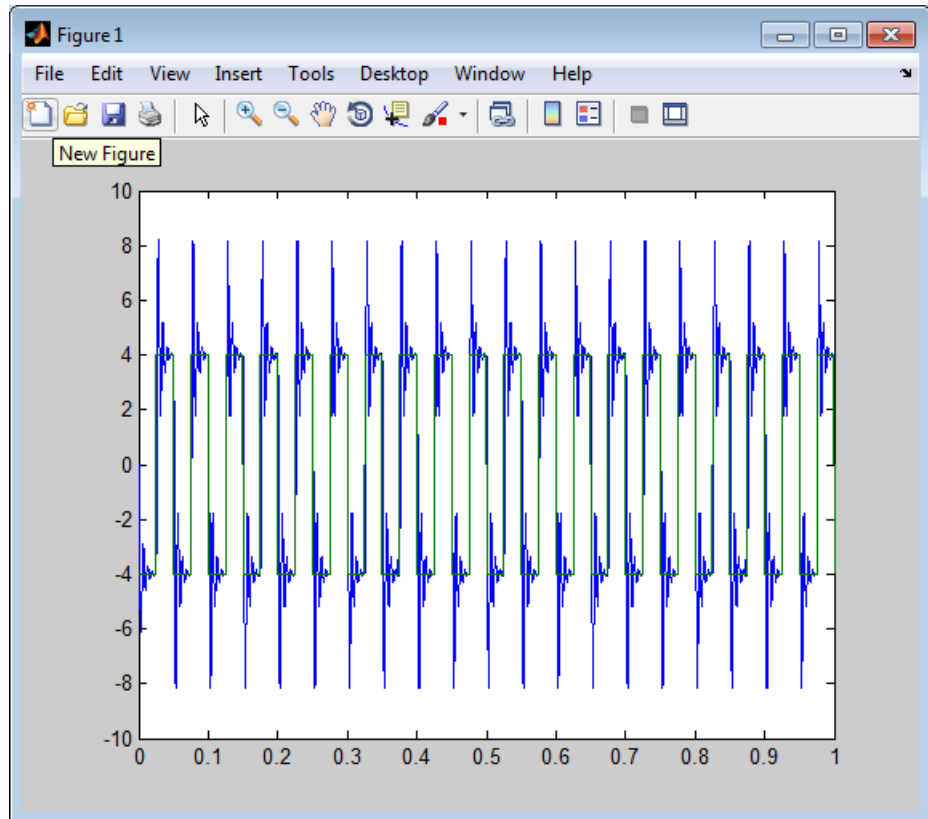
4 In the MATLAB window, type:

```
plot(tg.TimeLog,tg.TETLog)
```

Values for the task execution time (TET) log are uploaded to the host computer from the target computer. To upload part of the logs, see the target object method xpctarget.xpc.getlog.

The TET plot shown is the result of a real-time run.
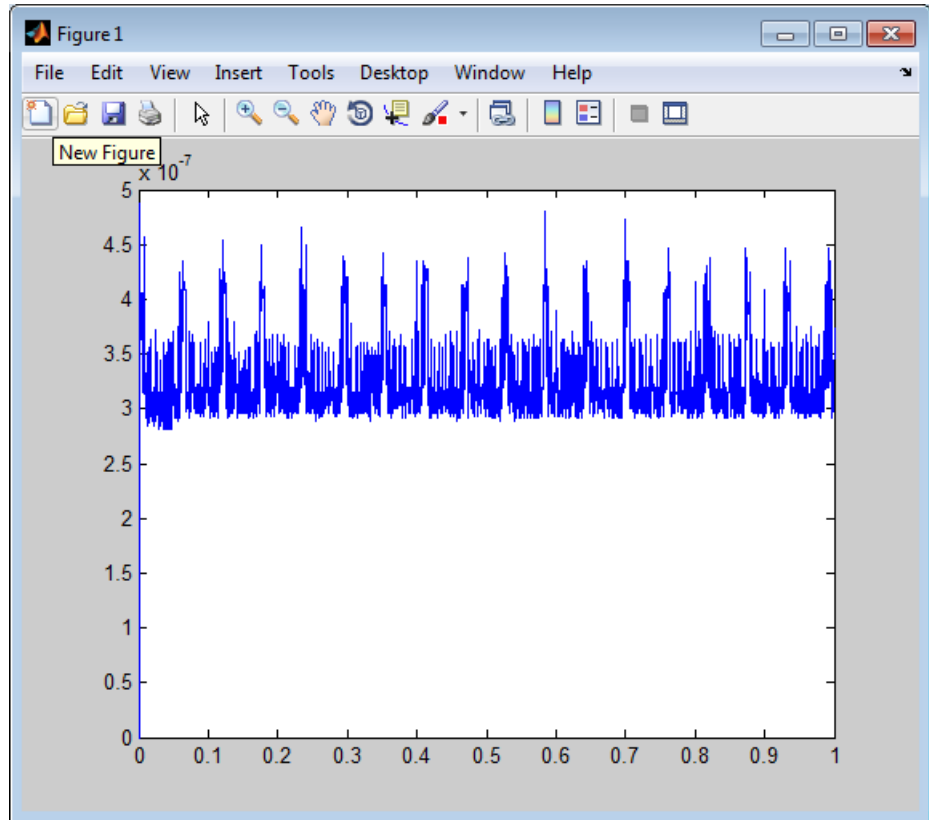
The TET is the time to calculate the signal values for the model during each sample interval. If you have subsystems that run only under certain circumstances, plotting the TET shows when subsystems were executed and the additional CPU time required for those executions.

5 In the MATLAB window, type:

```
tg.AvgTET
```

The MATLAB interface displays information about the average task execution time, for example:

```
ans =
   5.7528e-006
```

The percentage of CPU performance is the average TET divided by the sample time.

Each outport has an associated column vector in `tg.OutputLog`. You can access the data that corresponds to a particular outport by specifying the column vector for that outport. For example, to access the data that corresponds to Outport 2, use `tg.outputlog(:,2)`.

Configure File Scopes Using MATLAB Language

This procedure shows how to trace signals with file scopes using the Simulink model xpcosc as an example. You must have already built and downloaded the target application for this model. It also assumes that you have a serial communication connection.

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

---

1 Create a target object tg that represents the target application. Type:

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

2 To get a list of signals, type:

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

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

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

For more information, see "Monitor Signals Using MATLAB Language" on page 5-8.

3 Start running your target application. Type:

```
tg.start
```

The target computer displays the following message:

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

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

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

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

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

```
xPC Scope Object
   Application       = xpcosc
   ScopeId           = 2
   Status            = Interrupted
   Type              = File
   NumSamples        = 250
   NumPrePostSamples = 0
   Decimation        = 1
   TriggerMode       = FreeRun
   TriggerScope      = 2
   TriggerSample     = 0
   TriggerSignal     = -1
   TriggerLevel      = 0.000000
   TriggerSlope      = Either
   ShowSignals       = off
   FileName          = unset
   Mode              = Lazy
   WriteSize         = 512
   AutoRestart       = off
   DynamicFileName   = off
   MaxWriteFileSize  = 536870912
```

No name is initially assigned to FileName. After you start the scope, xPC Target assigns a name for the file to acquire the signal data. This name

typically consists of the scope object name, `ScopeId`, and the beginning letters of the first signal added to the scope.

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

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

The target computer displays the following messages:

```
Scope: 2, signal 4 added
Scope: 2, signal 5 added
```

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

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

```
sc2.start
```

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

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

```
Mode                = Lazy
WriteSize           = 512
AutoRestart         = off
DynamicFileName     = off
MaxWriteFileSize    = 536870912
```

8 Stop the scope. Type:

```
sc2.stop
```

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

```
tg.stop
```

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

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

To access the contents of the signal data file that the file scope creates, use the xPC Target file system object (`xpctarget.fs`) from the host computer MATLAB window. To view or examine the signal data, you can use the `readxpcfile` utility with the `plot` function. For further details on the `xpctarget.fs` file system object and the `readxpcfile` utility, see "Using xpctarget.fs Objects" on page 8-10.

Log Signals Using a Web Browser

When you stop the model execution, you see another section of the Web browser interface where you can download logging data. This data is in comma-separated value (CSV) format. This format can be read by most spreadsheet programs and by the MATLAB interface using the dlmread function.

You see this section of the Web browser interface only if you have enabled data logging. Buttons become available 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.

To perform data logging, you must complete the following setup:

1 Before you build the target application, add Outport blocks to your Simulink model. In the **Data Import/Export** pane of the Configuration Parameters dialog box, select the **Save to workspace** check box . See "Configure Simulation Parameters".

2 To plot the task execution time, in the **xPC Target options** pane of the Configuration Parameters dialog box, verify that the **Log Task Execution Time** check box is selected. This check box is selected by default. See "Add xPC Target Scope Block".

3 In the **xPC Target options** pane of the Configuration Parameters dialog box, set **Signal logging buffer size in doubles** to a value large enough to accommodate the logged signals. The default is 100000. If the default buffer size is not large enough, approximate the size using this formula:

```
Buffer size in doubles = 90% * Memory / sizeof(double)
```

Memory is the number of bytes available on the target computer after the kernel starts. It is displayed in the upper-left corner of the target computer screen. For example, for a Memory value of 2044MB, set **Signal logging buffer size in doubles** to 255500000.

The xPC Target software calculates the number of samples N for a signal as the value of **Signal logging buffer size in doubles** divided by the

number of logged signals (1 time, 1 task execution time ([TET]), number of outputs, number of states). The scopes copy the last N samples from the log buffer to the target object logs (`tg.TimeLog`, `tg.OutputLog`, `tg.StateLog`, and `tg.TETLog`).

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.

Parameter Tuning Basics

By default, the xPC Target software lets you change parameters in your target application while it is running in real time.

Note Some parameters are not observable. See "Nonobservable Signals and Parameters" on page 5-140.

You can improve overall efficiency by inlining parameters. The xPC Target product supports the Simulink Coder inline parameters functionality. (For more information about inlined parameters, see the Simulink Coder documentation.)

By default, inlined parameters are nontunable. If you want to make some of the inlined parameters tunable, do so through the Model Parameter Configuration dialog box (see "Configure Model to Tune Inlined Parameters" on page 5-131).

Tune Parameters Using xPC Target Explorer

You can use xPC Target Explorer to change parameters in your target application while it is running in real time or between runs. You do not need to rebuild the Simulink model, set the Simulink interface to external mode, or connect the Simulink interface with the target application.

This procedure uses model xpcosc. You must have already completed the setup tasks in "Create Host Scopes Using xPC Target Explorer" on page 5-58.

1 Select the target application in the **Applications** pane (for example, **xpcosc**).

2 To start execution, click the target application and then click the Start icon ▶ on the toolbar.

3 To start **Scope 1**, click **Scope 1** in the **Scopes** pane and then click the Start Scope icon 🔳 on the toolbar.
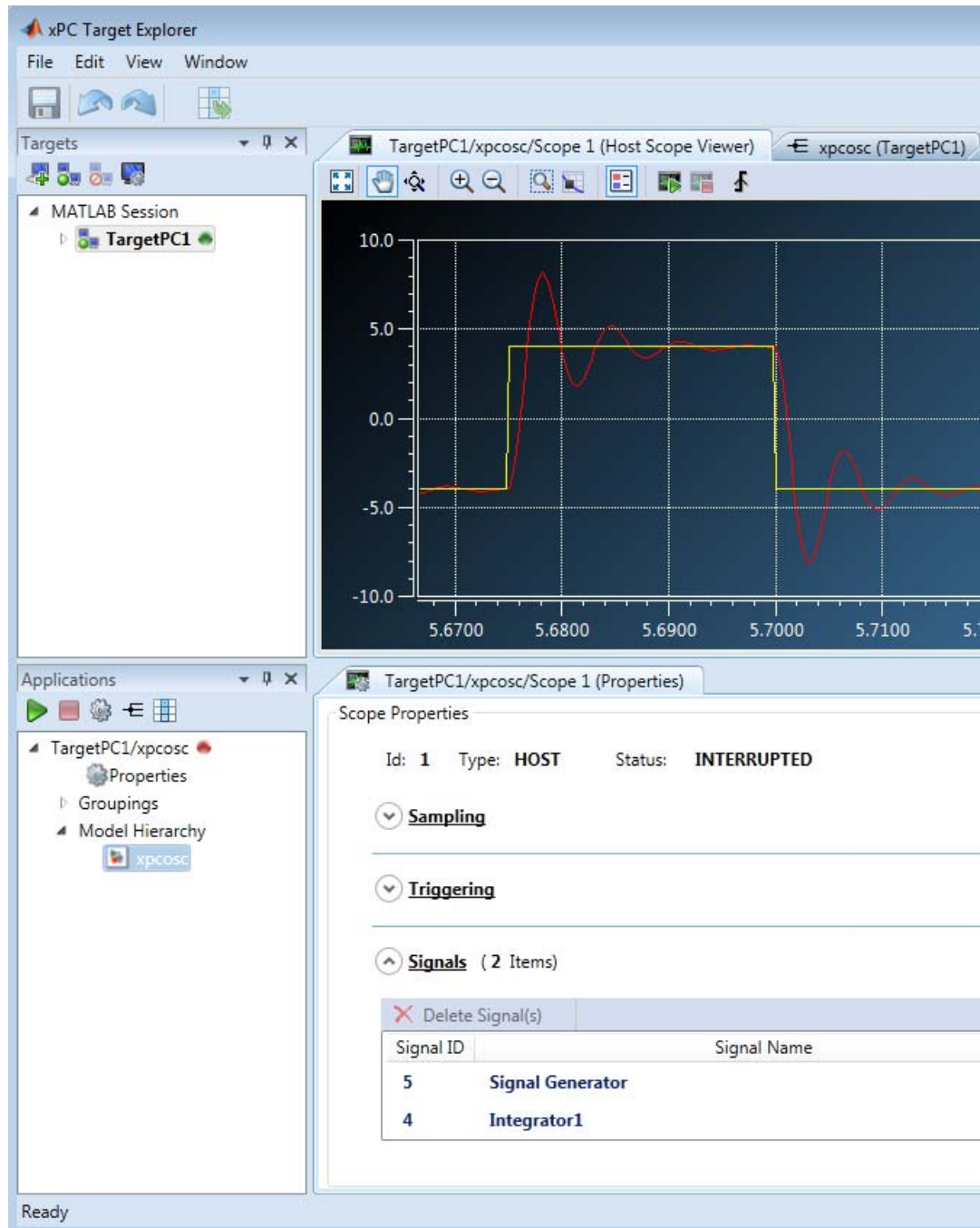
4 In the **Target Applications** pane, expand both the target application node and node **Model Hierarchy**.

5 Select the model node, and then click the View Parameters icon on the toolbar.

The Parameters workspace opens, showing a table of parameters with properties and actions.

6 Click the arrow next to the Gain for block Gain1. The **Values** text box opens, containing the initial value 400.

7 Type 100 into the text box, and then click outside the box.

To revert the Gain for block Gain1 to its previous value, click the Revert icon .

8 Click the **Apply parameter value(s) changes** icon .

The xPC Target Explorer window looks like this figure.
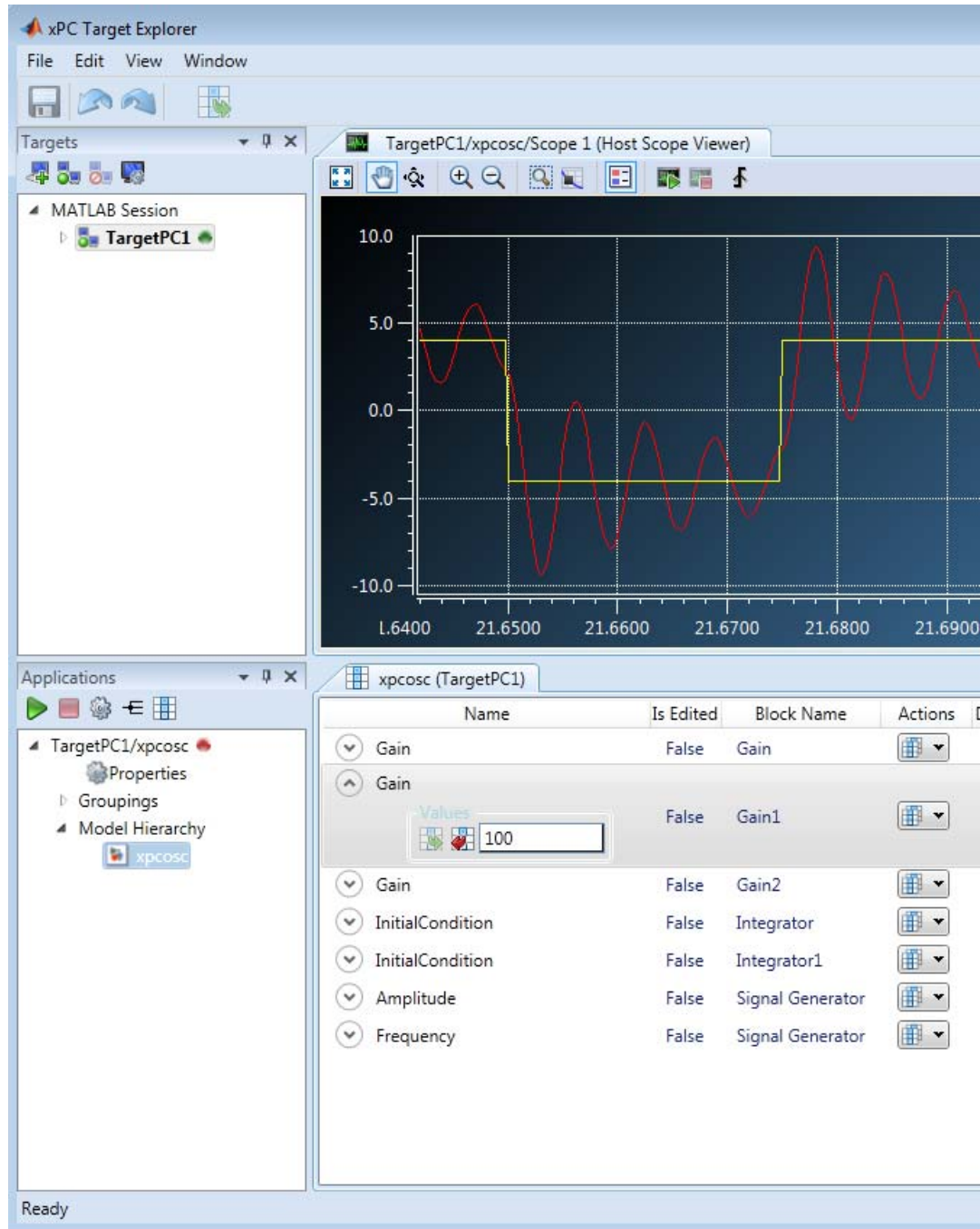
9 To stop **Scope 1**, click **Scope 1** in the **Scopes** pane, and then click the Stop Scope icon ▨ on the toolbar.

10 To stop execution, click the target application, and then click the Stop icon ▨ on the toolbar.

To group parameters, see "Create Parameter Groups Using xPC Target Explorer" on page 5-118

Create Parameter Groups Using xPC Target Explorer

When testing a complex model composed of many reference models, you must tune parameters from multiple parts and levels of the model. To do so, create a parameter group.

This procedure uses the model xpcosc as an example. You must have already completed the following setup:

1 Built and downloaded the target application to the target computer using Simulink (⊞ on the toolbar).

2 Run xPC Target Explorer (command xpcexplr).

3 Connected to the target computer in the **Targets** pane (⊙ on the toolbar).

To create a parameter group:

1 In the **Target Applications** pane, expand the target application node, and then right-click the **Groupings** node.

2 Click **New Parameter Group**.

3 In the Add New Parameter Group Item dialog box, enter a name in the **Name** text box (for example, **ParamGroup1.par**). In the **Location** text box, enter a folder for the group file.

4 Click **OK**. A new parameter group appears, along with its Parameter Group workspace.

5 In the **Target Applications** pane, expand both the target application node and the node **Model Hierarchy**.

6 Select the model node, and then click the View Parameters icon ⊞ on the toolbar.

The Parameters workspace opens, showing a table of parameters with properties and actions.

7 In the Parameter Groups workspace, to add parameter Amplitude to **ParamGroup1.par**, click the down arrow next to the Parameters Grouping icon in its Actions column.

A list of parameter groups appears, including **ParamGroup1.par**.

8 Click the Add Parameter icon next to **ParamGroup1.par**.

9 Add parameter Frequency to **ParamGroup1.par** in the same way.

10 Click in the Parameter Group workspace, and then click the Save icon on the toolbar.
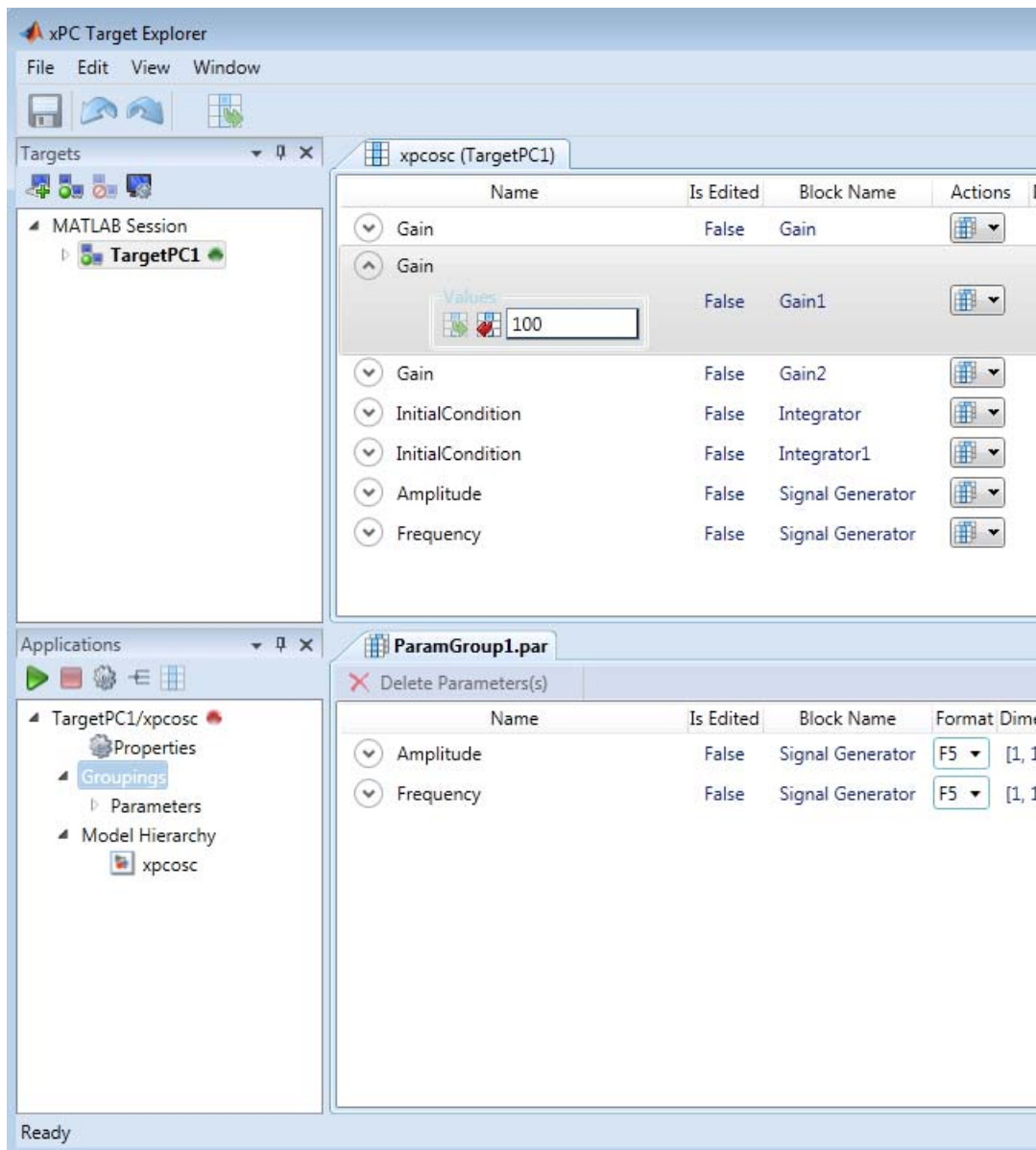
- To tune individual parameters in the selected group, see "Tune Parameters Using xPC Target Explorer" on page 5-113.

- To make both workspaces visible at the same time, click and hold the tab for one workspace and drag it down until the following icon appears in the middle of the dialog box: . Continue to drag until the cursor reaches the required quadrant, and then release the mouse button.

Tune Parameters Using MATLAB Language

You use the MATLAB functions to change block parameters. With these functions, you do not need to set the Simulink interface to external mode. You also do not need to connect the Simulink interface with the target application.

You can download parameters to the target application while it is running or between runs. You can change parameters in your target application without rebuilding the Simulink model and, if required, change them back to their original values. using xPC Target functions.

This procedure uses the Simulink model xpcosc as an example. You must have already created and downloaded the target application for that model and assigned tg to the target computer.

1 In the MATLAB window, type:

```
tg.start
```

The target computer displays the following message:

```
System: execution started (sample time: 0.001000)
```

2 Display a list of parameters. Type:

```
tg.ShowParameters='on'
```

The ShowParameters command displays a list of properties for the target object.

```
ShowParameters = on

Parameters =
```

INDEX	VALUE	TYPE	SIZE	PARAMETER NAME	BLOCK NAME
0	1000000	DOUBLE	Scalar	Gain	Gain
1	400	DOUBLE	Scalar	Gain	Gain1
2	1000000	DOUBLE	Scalar	Gain	Gain2

				Initial	
3	0	DOUBLE	Scalar	Condition	Integrator
4	0	DOUBLE	Scalar	Initial Condition	Integrator1
5	4	DOUBLE	Scalar	Amplitude	Signal Generator
6	20	DOUBLE	Scalar	Frequency	Signal Generator

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

```
pt = tg.setparam(1,800)
```

The `setparam` method returns a structure that stores the parameter index, the previous value, and the new value.

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

```
pt =
parIndexVec: 1
OldValues: 400
NewValues: 800
```

The target application runs. The plot frame updates the signals for the active scopes.

4 Stop the target application. In the MATLAB window, type:

```
tg.stop
```

The target application on the target computer stops running. The target computer displays messages like the following:

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

5 To reset to the previous values, type:

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

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

Note Method names are case sensitive and must be complete. Property names are not case sensitive and do not need to be complete, as long as they are unique.

Tune Parameters Using Simulink External Mode

You use Simulink external mode to connect your Simulink block diagram to your target application. The block diagram becomes a graphical user interface to your target application. You set up the Simulink interface in external mode to establish a communication channel between your Simulink block window and your target application.

In Simulink external mode, wherever you change parameters in the Simulink block diagram, the Simulink software downloads those parameters to the target application while it is running. You can change parameters in your program without rebuilding the Simulink model to create a new target application.

After you download your target application to the target computer, you can connect your Simulink model to the target application. This procedure uses the Simulink model xpcosc as an example. You must have already created and downloaded the target application for that model.

1 In the Simulink window, click **Simulation > Mode > External**. A check mark appears next to the menu item **External**, and Simulink external mode is activated.

2 Click the Connect To Target icon ⊙ on the toolbar. The current Simulink model parameters are downloaded from the host computer to your target application.

3 Click the Run icon ▶ on the toolbar.

The target application begins running on the target computer, and the target computer displays the following message:

```
System: execution started (sample time: 0.000250)
```

4 From the Simulation block diagram, double-click the block labeled **Gain1**

5 In the Block Parameters: Gain1 parameter dialog box, the **Gain** text box, enter 800. Click **OK**.

As soon as you change a model parameter and click **OK**, the changed parameters in the model are downloaded to the target application.

6 From the **Simulation** menu, click **Disconnect from Target**.

The Simulink model is disconnected from the target application. If you then change a block parameter in the Simulink model, the target application does not change.

7 In the MATLAB window, type:

```
tg=xpc('TargetPC1')
tg.stop
```

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

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

Tune Parameters Using a Web Browser

The **Parameters** pane displays the tunable parameters in your model. Row and column indices for vector/matrix parameters are also shown.

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

1 In the left frame, click **Parameters**. The browser loads the **Parameter List** pane into the right frame.

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

If the parameter is a vector or matrix, click a button to open another page that displays the vector or matrix. You can then edit the parameter.

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

The new parameter values are uploaded to the target application.

Save and Reload Parameters Using MATLAB Language

After you have a set of target application parameter values that you are satisfied with, you can save those values to a file on the target computer. You can then later reload these saved parameter values to the same target application.

You can save parameters from your target application while the target application is running or between runs. You can save and restore parameters in your target application without rebuilding the Simulink model. 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.

You save and restore parameters with the target object methods `saveparamset` and `loadparamset`.

Requirements:

- You have a target application object named `tg`.

- You have assigned `tg` to the target computer.

- You have downloaded a target application to the target computer.

- You have parameters to save. For more information, see:
  - "Tune Parameters Using MATLAB Language" on page 5-122
  - "Tune Parameters Using Simulink External Mode" on page 5-125
  - "Tune Parameters Using a Web Browser" on page 5-127

Save the Current Set of Target Application Parameters

To save a set of parameters to a target application, use the `saveparamset` method. The target application can be stopped or running.

1 Identify the set of parameter values that you want to save.

2 Select a descriptive file name to contain these values. For example, use the model name in the file name.

3 In the MATLAB window, type either

```
tg.saveparamset('xpc_osc4_param1')
```

The xPC Target software creates a file named `xpcosc4_param1` in the current folder of the target computer, for example, `C:\xpcosc4_param1`.

- To restore parameter values to a target application, see "Load Saved Parameters to a Target Application" on page 5-129.

- To list the parameters and values stored in the parameter file, see "List the Values of Parameters Stored in a File" on page 5-130.

Load 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 "Save the Current Set of Target Application Parameters" on page 5-128).

1 From the collection of parameter value files on the target computer, select the one that contains the parameter values you want to load.

2 In the MATLAB window, type:

```
tg.loadparamset('xpc_osc4_param1')
```

The xPC Target software loads the parameter values into the target application.

5-129

Tip

- To load the parameter set automatically during startup, see "Load a parameter set from a file on the designated target file system".

- To list the parameters and values stored in the parameter file, see "List the Values of Parameters Stored in a File" on page 5-130.

List the Values of Parameters Stored in a File

To list parameters and their values, load the file for a target application, and then turn on the ShowParameters target object property.

You must have a parameters file saved from an earlier run of saveparamset (see "Save the Current Set of Target Application Parameters" on page 5-128).

1 Stop the target application. In the MATLAB window, type:

```
tg.stop
```

2 Load the parameter file. Type:

```
tg.loadparamset('xpc_osc4_param1');
```

3 Display a list of parameters. Type:

```
tg.ShowParameters='on'
```

The MATLAB window displays a list of parameters and their values for the target object.

Configure Model to Tune Inlined Parameters

This procedure describes how you can globally inline parameters for a model, and then specify which of these parameters you still want to be tunable.

Note You cannot tune inlined parameters that are structures.

The following procedure uses the Simulink model `xpcosc` as an example.

1 In the MATLAB Command Window, type `xpcosc`. The model is displayed in the Simulink window.

2 Select the blocks of the parameters that you want to make tunable. For example, this procedure makes the signal generator's amplitude parameter tunable. Use the variable `A` to represent the amplitude.

3 Double-click the Signal Generator block, and then enter `A` for the `Amplitude` parameter. Click **OK**.

4 In the MATLAB Command Window, assign a constant to that variable. For example, type:

```
A = 4
```

The value is displayed in the MATLAB workspace.

5 From the Simulink window, click **Simulation > Model Configuration Parameters**.

6 In the Configuration Parameters dialog box, select the **Signals and Parameters** node under **Optimization**.

7 In the right pane, select the **Inline parameters** check box.

8 Click **Configure**.

The Model Parameter Configuration dialog box opens. The MATLAB workspace contains the constant you assigned to `A`.

9 Select the line that contains your constant. Click **Add to table**.
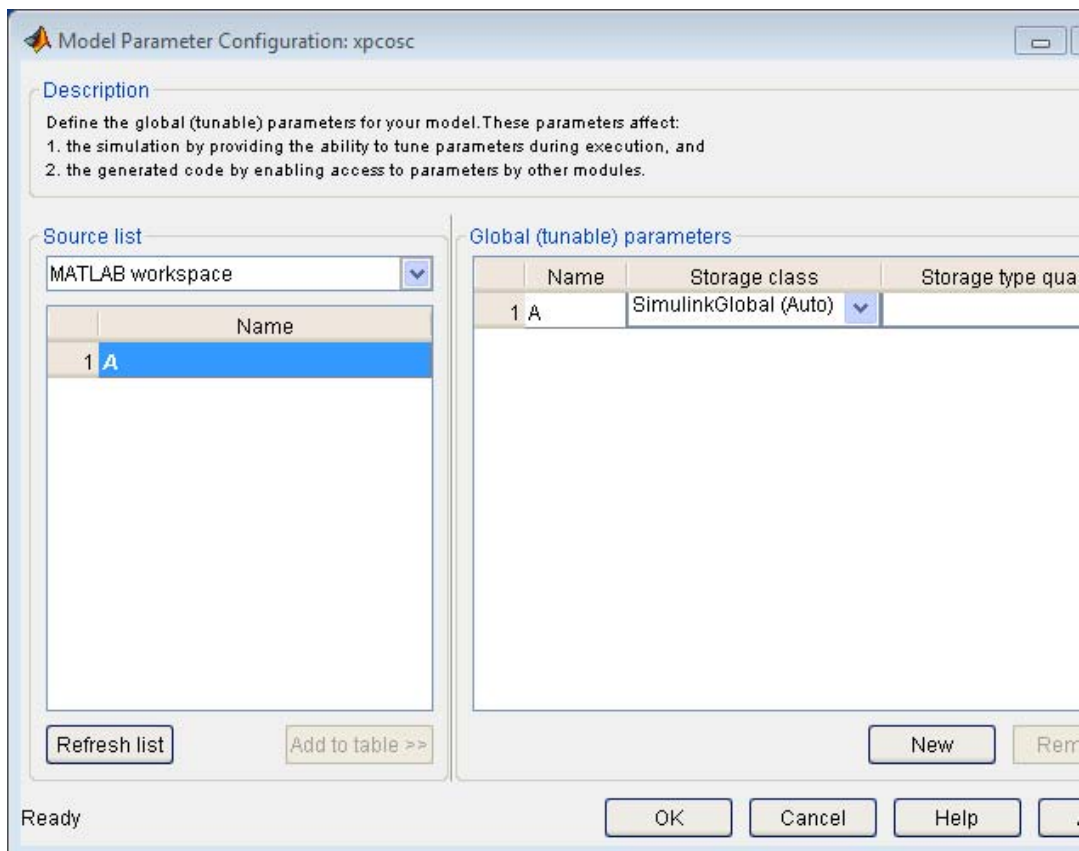
Add the remaining global parameters that you want to tune.

10 Click **Apply**, and then click **OK**.

11 In the Configuration Parameters dialog box, click **Apply**, and then **OK**.

12 Save the model. For example, save it as xpc_osc5.

13 Build and download the model to your target computer.

14 To tune inline parameters, see either:

- "Tune Inlined Parameters Using xPC Target Explorer" on page 5-134

- "Tune Inlined Parameters Using MATLAB Language" on page 5-139

Tune Inlined Parameters Using xPC Target Explorer

This procedure describes how you can tune inlined parameters through the xPC Target Explorer.

The procedure uses the model xpc_osc5 from "Configure Model to Tune Inlined Parameters" on page 5-131 as an example. You must have already completed the setup tasks in "Create Host Scopes Using xPC Target Explorer" on page 5-58.

1 Select the target application in the **Applications** pane (for example, **xpc_osc5**).

2 To start execution, click the target application, and then click the Start icon ▶ on the toolbar.

3 To start **Scope 1**, click **Scope 1** in the **Scopes** pane, and then click the Start Scope icon on the toolbar.

4 In the **Target Applications** pane, expand both the target application node and the **Model Hierarchy** node.

5 Select the model node, and then click the View Parameters icon on the toolbar. The Parameters workspace opens, showing a table of parameters with properties and actions.

6 Click the arrow next to parameter A for block Model Parameters. The **Values** text box opens, containing the initial value 4.

7 Type 2 into the text box, and then click outside the box.

To revert parameter A for block Model Parameters to its previous value, click the Revert icon .

8 Click the **Apply parameter value(s) changes** icon .

The dialog looks like this figure.
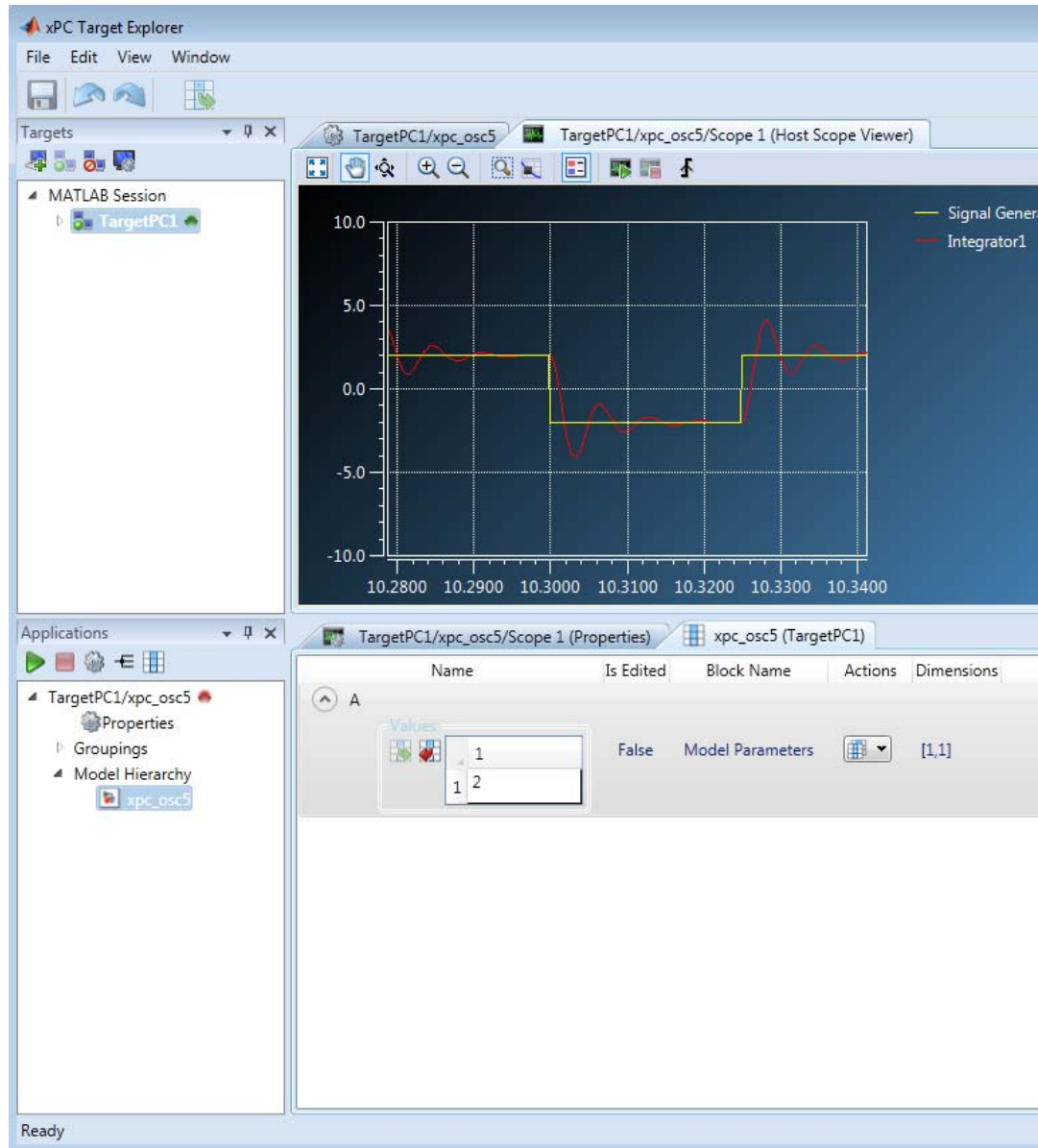
9 To stop **Scope 1**, click **Scope 1** in the **Scopes** pane, and then click the Stop Scope icon ![icon] on the toolbar.

10 To stop execution, click the target application, and then click the Stop icon ![icon] on the toolbar.

Tune Inlined Parameters Using MATLAB Language

This procedure describes how you can tune inlined parameters through the MATLAB interface. You must have already built and downloaded the model from the topic "Configure Model to Tune Inlined Parameters" on page 5-131 to the target computer. The model must already be running.

You can tune inlined parameters using a parameter ID.

- Use the `getparamid` function to get the ID of the inlined parameter that you want to tune. For the `block_name` parameter, leave a blank (`''`).

- Use the `setparam` function to set the new value for the inlined parameter.

1 Save the following code in a MATLAB file. For example, `change_inlineA`.

```
tg=xpc; %Create xPC Target object
pid=tg.getparamid('','A'); %Get parameter ID of A
if isempty(pid) %Check value of pid.
    error('Could not find A');
end
tg.setparam(pid,100); %If pid is valid, set parameter value.
```

2 Execute that MATLAB file. Type:

```
change_inlineA
```

3 To see the new parameter value, type:

```
tg.showparameters='on'
```

The `tg` object information is displayed, including the parameter lines:

```
NumParameters = 1

ShowParameters = on

Parameters = INDEX  VALUE  TYPE   SIZE   PARAMETER NAME  BLOCK
NAME

              0       100  DOUBLE Scalar A
```

Nonobservable Signals and Parameters

Observable signals are those signals that you can monitor, trace, and log. Nonobservable signals are those signals that exist in the target application, but are not observable from the host computer.

You cannot observe the following types of signals:

- Virtual or bus signals (including signals from bus and virtual blocks). You can access these signals from nonvirtual source blocks.

  To observe a virtual signal, add a Gain block with gain 1.0 (unit gain) and observe its output.

  To observe a virtual bus, add a Gain block with unit gain to each individual signal.

- Signals that you have optimized with block reduction optimization. You can access these signals by making them test points.
- Signals of complex or multiword data types.

Observable parameters are those parameters you can tune. Nonobservable parameters are those parameters that exist in the target application, but are not tunable from the host computer. You cannot observe the parameters of complex or multiword data types.

Execution Modes

Execution Modes

Interrupt mode is the default real-time execution mode for the xPC Target kernel. In certain conditions, you might want to change the real-time execution mode to polling mode. A good understanding of interrupt and polling modes will help you to use them effectively, and to decide under which circumstances it makes sense for you to switch to the polling mode.

A third execution mode, freerun, is also available. In this mode, the target application thread does not wait for the timer and the kernel runs the application as fast as possible. The time between each execution might vary if the target application has conditional code. The three execution modes are mutually exclusive. For a description of how to use the freerun mode, see "Set Configuration Parameters".

Interrupt Mode

Interrupt mode is the default real-time execution mode for the kernel. This mode provides the greatest flexibility and is the mode you should choose for applications that execute at the given base sample time without overloading the CPU.

The scheduler implements real-time single-tasking and multitasking execution of single-rate or multirate systems, including asynchronous events (interrupts). Additionally, background tasks like host-target communication or updating the target screen run in parallel with sample-time-based model tasks. This allows you to interact with the target system while the target application is executing in real time at high sample rates. This is made possible by an interrupt-driven real-time scheduler that is responsible for executing the various tasks according to their priority. The base sample time task can interrupt other tasks (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 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 µs 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 µs is introduced for the interrupt controller.

- CPU hardware latency — Modern CPUs try to predict the next couple of instructions, including branches, by the use of instruction pipelines. If an interrupt occurs, the prediction fails and the pipeline has to be fully reloaded. This process introduces an additional latency. Additionally, because of interrupts, cache misses will occur.

- Interrupt handler entry and exit latency — Because an interrupt can stop the currently executing task at an arbitrary instruction and the interrupted task has to resume when the interrupting task completes execution, its state has to be saved and restored accordingly. This includes saving CPU data and address registers, including the stack pointer. In the case that the interrupted task executed floating-point unit (FPU) operations, the FPU stack has to be saved as well (108 bytes on a Pentium CPU). This introduces additional latency.

- Interrupt handler content latency — If a background task has been executing for some time, say in a loop, its data will be available in the cache. When an interrupt occurs and the interrupt service handler is executed, the interrupt handler data could be purged from the cache, causing the CPU to reload it from slower RAM. This introduces additional latency. Because of its unpredictable nature, an interrupt generally reduces the optimal execution speed or introduces latency,.

The kernel in interrupt mode is close to optimal for executing code on a PC-compatible system. However, interrupt mode introduces an overall latency of about 8 μs. This is a significant amount of time when considering that a 1 GHz CPU can execute thousands of instructions within 8 μs. This time is equivalent to a Simulink model containing a hundred nontrivial blocks. Additionally, because lower priority tasks have to be serviced as well, at least 5% of headroom is required, which can cause additional cache misses and therefore nonoptimal execution speed.

Polling Mode

Polling mode for the kernel is designed to execute target applications at sample times close to the limit of the hardware (CPU). Using polling mode with high-speed and low-latency I/O boards and drivers allows you to achieve smaller sample times for applications that you cannot achieve using the interrupt mode of the xPC Target software.

Polling mode has two main applications:

- Control applications — Control applications of average model size and I/O complexity that are executed at very small sample times (Ts = 5 to 50 µs)

- 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 (Fs = 20 to 200 kHz)

Introducing Polling Mode

Polling mode for the kernel does not have the 8 µs of latency that interrupt mode does. This is because the kernel does not allow interrupts, so the CPU can use this extra time for executing model code.

Polling mode is sometimes seen as a "primitive" or "brute force" real-time execution scheme. Nevertheless, when a real-time application executes at a given base sample time in interrupt mode and overloads the CPU, switching to polling mode is often the only alternative to get the application to execute at the required sample time.

Polling means that the kernel waits in an empty while loop until the time at which the next model step has to be executed is reached. Then the next model step is executed. At least a counter implemented in hardware has to be accessible by the kernel in order to get a base reference for when the next model step execution has to commence. The kernel polls this hardware counter. If this hardware counter must be outside the CPU, e.g., in the chip set or even on an ISA or PCI board, the counter value can only be retrieved by an I/O or memory access cycle that again introduces latency. This latency usually eats up the freed-up time of polling mode. Fortunately, since the introduction of the Pentium CPU family from Intel, the CPU is equipped with a 64 bit counter on the CPU substrate itself, which commences counting at power-up time and counts up driven by the actual clock rate of the CPU.

Even a highly clocked CPU is not likely to lead to an overflow of a 64 bit counter (2^{64} * 1e-9 (1 GHz CPU) = 584 years). The Pentium counter comes with the following features:

- More precise measurements — Because the counter counts up with the CPU clock rate (~1 GHz nowadays), time measurements even in the microsecond range are very precise, leading to small real-time errors.

- Overflow handler not required— Because the counter is 64 bits wide, in practical use overflow does not occur, avoiding the CPU-time overhead of handling overflows.

- Minimal latency — The counter resides on the CPU. Reading the counter value can be done within one CPU cycle, introducing minimal latency.

The polling execution scheme does not depend on interrupt sources to notify the code to continue calculating the next model step. While this frees the CPU, it means that code that is part of the exclusively running polling loop is executed in real time – even components, which have so far been executed in background tasks. Because these background tasks are usually non-real-time tasks and can use a lot of CPU time, do not execute them. This is the main disadvantage of polling mode. To be efficient, only the target application's relevant parts should be executed. In the case of the xPC Target software, this is the code that represents the Simulink model itself.

Therefore, host-target communication and target display updating are disabled. Because polling mode reduces the features of the xPC Target product to a minimum, you should choose it only as the last possible alternative to reach the required base sample time for a given model. Therefore, do the following before you consider polling mode:

- The model is optimal concerning execution speed — First, you should run the model through the Simulink profiler to find possible speed optimizations using alternative blocks. If the model contains continuous states, the discretization of these states will reduce model complexity significantly, because a costly fixed-step integration algorithm can be avoided. If continuous states cannot be discretized, you should use the integration algorithm with the lowest order that still produces the required numerical results.

- Use the fastest available computer hardware — Use the CPU with the highest clock rate available for a given PC form factor. For the desktop form factor, this would mean a clock rate above 3 GHz; for a mobile application, e.g., using the PC/104 form factor, this would mean a clock rate above 1 GHz. Executing xpcbench at the MATLAB prompt gives a relative measure of CPU performance when running typical target applications.

- Use the lowest latency I/O hardware and drivers available — Many xPC Target applications communicate with hardware through I/O hardware over either an ISA or PCI bus. Because each register access to such I/O hardware introduces a comparably high latency time (~1 µs), the use of the lowest latency hardware/driver technology available is crucial.

- The base sample time is about 50 µs or less — The time additionally assigned to model code execution in polling mode is only about 8 µs. If the given base sample time of the target application exceeds about 50 µs, the possible percentage gain is rather small. Other optimization technologies might have a bigger impact on performance.

Setting the Polling Mode

Polling mode is an alternative to the default interrupt mode of the kernel. This means that the kernel on the bootable media created by the GUI allows running the target application in both modes without using another boot disk.

By default the target application executes in interrupt mode. To switch to polling mode, you need to pass an option to the **System target file** command.

The following example uses xpcosc.

1 In the Simulink window, select **Simulation > Model Configuration Parameters**.

The Configuration Parameters dialog box opens.

2 In the left pane, click the **Code Generation** node.

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

   -axpcCPUClockPoll=*CPUclockRateMHz*

You must assign the target computer CPU clock rate because the Pentium on-chip counter used for polling mode counts up with the CPU clock rate. If the clock rate is provided, the kernel can convert clock ticks to seconds and vice versa. If the required clock rate is not provided, the target application does not execute at the required base sample time. You can find out about the CPU clock rate of the target computer by rebooting the target computer and checking the screen output during BIOS execution time. The BIOS usually displays the CPU clock rate in MHz right after the target computer has been powered up.

For example, if your target computer is a 1.2 GHz AMD Athlon, specify the following option in the **TLC options** edit field:

```
-axpcCPUClockPoll=1200
```
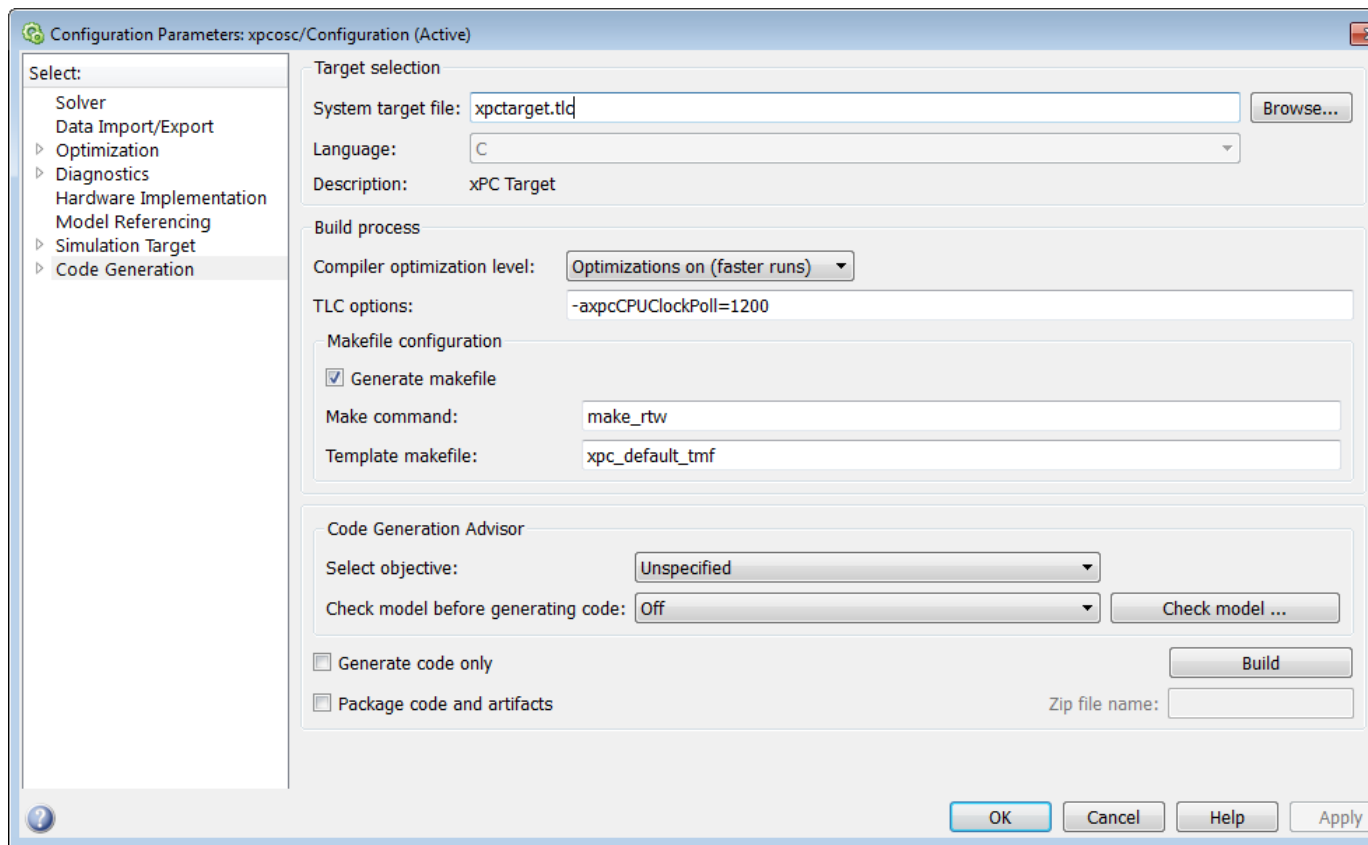
If you want to execute the target application in interrupt mode again, either remove the option or assign a CPU clock rate of 0 to the option:

```
-axpcCPUClockPoll=0
```

If you make a change to the **TLC options** field, you need to rebuild the target application for the change to take effect. Building the target application, downloading it, and preparing it for a run then work exactly the same way as they did with default interrupt mode.

After the download of the target application has succeeded, the target screen displays the mode, and if polling mode is activated, it additionally displays the defined CPU clock rate in MHz. This allows you to check the setting.

Restrictions Introduced by Polling Mode

As explained above, polling mode executes the Simulink-based target application in real time exclusively. While the target application is executing in polling mode, background tasks are inactive, including those for host-target communication, target screen updating, and UDP transfers. This is because interrupts of the target computer are fully disabled during the execution of the target application. On one hand, this improves polling performance; on the other hand, background tasks are not serviced.

The following topics list the 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 target-related commands on the host, to avoid displaying the same error message over and over again.

As a consequence, it is not possible to issue a `stop(tg)` command to stop the target application execution from the host side. The target application has to reach its set stop time for polling mode to be exited. You can use

```
tg.stoptime=x
```

before starting the execution, but once started the application executes until the stop time is reached.

Nevertheless, there is a way to stop the execution interactively before reaching the target application stop time. See "Controlling the Target Application" on page 6-13.

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. Selecting the **Graphics mode** check box in the **Target Properties** pane of xPC Target Explorer does not work. You should therefore clear the **Graphics mode** check box, producing text output only.

Session Time Does Not Advance During the Execution of the Target Application

Because interrupts are disabled during a run, the session time does not advance. The session time right before and after the run is therefore the same. This is a minor restriction and should not pose a problem.

The Only Rapid-Prototyping Feature Available Is Data Logging

Because host-target communication and target screen updating are disabled during the entire run, most of the common rapid-prototyping features of the xPC Target product are not available in polling mode:

- Parameter tuning
- Signal monitoring
- Scope objects
- Applications using the xPC Target APIs
- The Internet browser interface
- xpctargetspy and similar utilities

The only rapid-prototyping feature available is signal data logging, because signal data is acquired independently of the host, and logged data is retrieved only after the execution is stopped. Nevertheless, being able to log data allows gathering useful information about the behavior of the target application. Signal logging becomes a very important feature in polling mode.

Multirate Simulink Models Cannot Be Executed in Multitasking Mode on the Target Computer

Because of the polling mode execution scheme, executing Simulink-based target applications in multitasking mode is not possible. The modeling of function-call subsystems to handle asynchronous events (interrupts) is not possible either. This can be a hard restriction, especially for multirate systems. Multirate systems can be executed in single-tasking mode, but because of its scheme for sequential execution of subsystems with different rates, the CPU will most likely overload for the given base sample time. As an important consequence, polling mode is only a feasible alternative to interrupt mode if the model has a single rate or if it can be converted to a single-rate model. A single-rate model implies continuous states only, discrete states only, or mixed continuous and discrete states, if the continuous and discrete subsystems have the same rate. Use the Simulink **Display > Sample Time > Colors** feature to check for the single rate requirement. Additionally, to avoid a possible switch to multitasking mode, set the Tasking Mode property in the **Solver** pane of the Configuration Parameters dialog box to SingleTasking.

I/O Drivers Using Kernel Timing Information Cannot Be Used Within a Model

Some xPC Target drivers use timing information exported from the kernel to, for example, detect time-outs. Because the standard timing engine of the kernel does not run in polling mode, the required timing information is not passed back to the drivers. Therefore, in polling mode you cannot use drivers that import the header file time_xpcimport.h. This is a current restriction only. In a future version of polling mode, drivers will use the Pentium counter for timing instead.

Controlling the Target Application

As mentioned, you cannot 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, rapid-prototyping features other than signal logging are disabled. But because I/O driver blocks added to the model are fully functional, you can use I/O drivers to get to a minimal level of interactivity.

Stopping a target application using polling mode — You can use a low-latency digital input driver for the digital PCI board in your model, which reads in a single digital TTL signal. The signal is TTL low unless the model execution should be stopped, for which the signal changes to TTL high. You can connect the output port of the digital input driver block to the input port of a Stop simulation block, found in the standard Simulink block library. This stops the execution of the target application, depending on the state of the digital input signal. You can either use a hardware switch connected to the board-specific input pin or you can generate the signal by other means. For example, you could use another digital I/O board in the host machine and connect the two boards (one in the host, the other in the target) over a couple of wires. You could then use the Data Acquisition Toolbox™ product to drive the corresponding TTL output pin of the host board to stop the target application execution from within the MATLAB interface.

Generally, you can use the same software/hardware setup for passing other information back and forth during run time of the target application. However, you must implement features other than signal logging at the model level and therefore must minimize the additional latency introduced by the feature. For example, being able to interactively stop the target application

execution is paid for by the additional 1 μs latency required to read the digital signal over the digital I/O board. However, if you need to read digital inputs from the plant hardware anyway, and lines are available, you get the feature for free.

Polling Mode Benchmarks

This is preliminary information. Benchmarks were executed using a 1 GHz AMD Athlon machine. For more information about benchmarks, see xpcbench or type `help xpcbench` in the MATLAB Command Window.

- The minimum achievable base sample time for benchmark model Minimal is 1 μs with signal logging disabled and 2 μs with signal logging enabled.

- The minimum achievable base sample time for model f14 using an ode4 fixed-step integration algorithm is 4 μs with signal logging disabled and 5 μs with signal logging enabled.

A more realistic model that 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 μs with signal logging disabled and 12 μs 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 ~28 μs or a sample rate of ~33 kHz. For this application, the overall performance increase using polling mode is almost a factor of 3.

Polling Mode and Multicore Processors

If your target computer has multicore processors, enabling the multicore processor supports removes the following restrictions. Other restrictions still apply.

Tip For more on how to enable multicore processor support, see "Multicore Processor Configuration" on page 26-4.

- Host-target communication is now available during the execution of the target application.

- Target screen now updates during the execution of the target application.

- External interrupts are now allowed during the execution of the real-time model. This does not mean that you can trigger your model with an external interrupt.

- File scopes can now log data into a file on the target computer.

Execution Using MATLAB Scripts

An important part of the "Rapid Prototyping" and "Hardware in the Loop" workflows is preparing stress test and regression test scripts. The xPC Target product includes specialized MATLAB classes and functions for setting up the target environment, booting the target computer, loading and running the target application, and displaying and recording the results. You can do these tasks using MATLAB functions and target and scope class objects.

- Chapter 7, "Targets and Scopes in the MATLAB Interface"
- Chapter 8, "Logging Signal Data with FTP and File System Objects"

7

Targets and Scopes in the MATLAB Interface

Target Driver Objects

The xPC Target software uses a target object (of class `xpctarget.xpc`) to represent the target kernel and your target application. Use target object functions to run and control real-time applications on the target computer with scope objects to collect signal data.

An understanding of the target object properties and methods helps you to control and test your application on the target computer.

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

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

To create a target object:

1 Build a target application. The xPC Target software creates a target object during the build process.

2 Use the target object constructor function `xpctarget.xpc`. In the MATLAB Command window, type `tg = xpctarget.xpc`.

Target objects are of class `xpctarget.xpc Class`. A target object has associated properties and methods specific to that object. The target application object methods allow you to control a target application on the target computer from the host computer. You enter target application object methods in the MATLAB window on the host computer, or you can use MATLAB code scripts. To access the help for these methods from the command line, use the syntax:

```
help xpctarget.xpc/method_name
```

If you want to control the target application from the target computer, use target computer commands (see "Target Computer Command-Line Interface" on page 10-2).

Create Target Objects

To create a target object:

1 Build a target application. The xPC Target software creates a target object during the build process.

2 To create a single target object, or to create multiple target objects in your system, use the target object constructor function `xpctarget.xpc` with arguments. For example, the following creates a target object connected to the host through an RS-232 connection. In the MATLAB Command Window, type:

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

The resulting target object is `tg`.

Using this method clarifies which target object is associated with a particular target computer.

3 To check a connection between a host and a target, use the target function `xpctarget.xpc.targetping`. For example, type:

```
tg.targetping
```

4 To create a single target object, or to create the first of many targets in your system, use the target object constructor function `xpctarget.xpc` without arguments. For example, in the MATLAB Command Window, type:

```
tg = xpctarget.xpc
```

The resulting target object is `tg`.

Note If you use `xpctarget.xpc` without arguments to create a target object, use xPC Target Explorer to configure your target computer. Doing so clarifies which target object is associated with a particular target computer.

Display Target Object Properties

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

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

1 In the MATLAB window, type:

```
tg
```

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

The target object properties for `TimeLog`, `StateLog`, `OutputLog`, and `TETLog` are not updated at this time.

2 Type:

```
tg.start
```

The `Status` property changes from `stopped` to `running`. The log properties change to `Acquiring`.

For a list of target object properties with a description, see the target object function `xpctarget.xpc.get (target application object)`.

Set Target Object Property Values

You can change a target object property by using the xPC Target software `set` method or the dot notation on the host computer. (For limitations on target property changes to sample times, see "User Interaction".)

With the xPC Target software, you can use a function syntax or an object property syntax to change the target object properties. The syntax `set(target_object, property_name, new_property_value)` can be replaced by:

```
target_object.property_name = new_property_value
```

For example, to change the stop time for target object `tg`, in the MATLAB window, type one of the following:

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

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

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

Get Target Object Property Values

You can list a property value in the MATLAB window or assign that value to a MATLAB variable. With the xPC Target software, you can use either a function syntax or an object property syntax. The syntax `get(target_object, property_name)` can be replaced by:

`target_object.property_name`

For example, to access the stop time for target object `tg`, in the MATLAB window, type one of the following:

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

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

Signals are not target object properties. To get the value of the `Integrator1` signal from the model `xpcosc`, in the MATLAB window, type one of the following:

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

`0` is the signal index.

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.

Use Target Object Methods

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, in lowercase. For example, to add a target scope with a scope index of 1, in the MATLAB window, type one of the following:

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

Target Scope Objects

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

The xPC Target software uses scopes and scope objects as an alternative to using Simulink scopes and external mode. A scope can exist as part of a Simulink model system or outside a model system.

- A scope that is part of a Simulink model system is a scope block. You add an xPC Target scope block to the model, build an application from that model, and download that application to the target computer.

- A scope that is outside a model is not a scope block. For example, if you create a scope with the `xpctarget.xpc.addscope` method, that scope is not part of a model system. After the model has been downloaded and initialized, you add this scope to the model.

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. In the latter case, the scope might acquire samples at irregular intervals.

A scope that is not part of a model executes at the base sample time of the model. Therefore, 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 acquires five identical samples for this signal, and then the next five identical samples, and so on.

Understanding the structure of scope objects helps you to use the MATLAB command-line interface to view and collect signal data. A scope object on the host computer represents a scope on the target computer. You use scope objects to observe the signals from your target application during a real-time run or analyze the data after the run is finished.

To create a scope object:

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

- Use the target object method `xpctarget.xpc.addscope` to create a scope, create a scope object, and assign the scope properties to the scope object.

Upon creation, the xPC Target software assigns the required scope object class for the scope type:

- Target scopes — `xpctarget.xpcsctg Class`, created by calling `xpctarget.xpc.getscope` with scope type `target`.

- Host scopes — `xpctarget.xpcschost Class`, created by calling `xpctarget.xpc.getscope` with scope type `host`.

- File scopes – `xpctarget.xpcfs Class`, created by calling `xpctarget.xpc.getscope` with scope type `file`.

A scope object has associated properties and methods specific to that scope type. These scope types are based on a common type, `xpctarget.xpcsc Class`, that encompasses their common properties and methods. If you create multiple scopes of different types for one model and combine those scopes, for example, into a scope vector, the xPC Target software creates this object.

The scope object methods allow you to control scopes on your target computer.

If you want to control the target application from the target computer, use target computer commands (see "Target Computer Command-Line Interface" on page 10-2).

Display Scope Object Properties for One Scope

To list the properties of a single scope object, sc1, in the MATLAB window, type one of the following:

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

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

The current scope properties are uploaded to the host computer. 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 of type host store data in the properties scope_object.Time and scope_object.Data.

For a list of target object properties with a description, see the target function xpctarget.xpc.get (target application object).

Display Scope Object Properties for the Current Scopes

To list the properties of the current scope objects associated with the target object `tg`, in the MATLAB window, type one of the following:

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

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

Alternatively, type one of the following:

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

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

For a list of target object properties with a description, see the target function `xpctarget.xpc.get (target application object)`.

Set Scope Property Values

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

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

For example, to change the trigger mode for the scope object `sc1`, in the MATLAB window, type one of the following:

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

You cannot use dot notation to set vector object properties. To assign properties to a vector of scopes, use the `set` method. For example, assume you have a variable `sc12` for two scopes, 1 and 2. To set the `NumSamples` property of these scopes to 300, in the MATLAB window, type the following:

```
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.

Get Scope Property Values

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

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

`scope_object_vector(index_vector).property_name`

For example, to assign the number of samples from the scope object `sc1`, in the MATLAB window, type one of the following:

```
numsamples = sc1.NumSamples
numsamples = sc1.get('NumSamples')
numsamples = get(sc1,'NumSamples')
```

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 the following:

```
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.

Use Scope Object Methods

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

```
scope_object.method_name(argument_list)
scope_object_vector(index_vector).method_name(argument_list)
```

Unlike properties, for which partial but unambiguous names are permitted, enter method names in full, in lowercase. For example, to add signals to the first scope in a vector of all scopes, in the MATLAB window, type one of the following:

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

Acquire Signal Data with File Scopes

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

Note Remember to start your scope to acquire signal data.

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

1 In the MATLAB window, type:

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

The xPC Target software creates a file scope for the application.

2 To add signal 4, type:

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

3 To start the scope, type:

```
sc.start
```

4 To start the target application, type:

```
tg.start
```

The xPC Target software adds signal 4 to the file scope. When you start the scope and application, the scope saves the signal data for signal 4 to a file, by default named `C:\data.dat`.

For more information on file scopes, see "Configure File Scope (xPC) Blocks" on page 5-78.

If you want to acquire signal data into multiple files, see "Acquire Signal Data into Dynamically Named Files" on page 7-16.

Acquire Signal Data into Dynamically Named Files

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

Note Remember to start your scope to acquire signal data.

For example, configure a file scope named sc to the application with the following characteristics:

- Logs signal data into up to nine files whose sizes do not exceed 4096 bytes.

- Creates files whose names contain the string file_.dat.

- Contains signal 4.

1 In the MATLAB window, type:

```
tg.StopTime=-1;
```

This parameter directs the target application to run indefinitely.

2 To add a file scope, type:

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

3 To enable the file scope to create multiple log files, type:

```
sc.DynamicFileName='on';
```

Enable this setting to enable logging to multiple files.

4 To enable file scopes to collect data up to the number of samples, and then start over again, type:

```
sc.AutoRestart='on';
```

Use this setting for the creation of multiple log files.

5 To limit each log file size to `4096`, type:

```
sc.MaxWriteFileSize=4096;
```

You must use this property. Set `MaxWriteFileSize` to a multiple of the `WriteSize` property.

6 To enable the file scope to create multiple log files with the same name pattern, type:

```
sc.Filename='file_<%>.dat';
```

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

7 To add signal 4 to the file scope, type:

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

8 To start the scope, type

```
sc.start
```

9 To start the target application, type

```
tg.start
```

The software creates a log file named `file_1.dat` and writes data to that file. When the size of `file_1.dat` reaches 4096 bytes (value of `MaxWriteFileSize`), the software closes the file and creates `file_2.dat`. When its size reaches 4096 bytes, the software closes it and creates `file_3.dat`, and so on.

The software repeats this sequence until it fills the last log file, `file_9.dat`. If the target application continues to run and collect data after `file_9.dat`, the software reopens `file_1.dat` and overwrites the existing contents. It cycles through the other log files sequentially. If you do not retrieve the data from existing files before they are overwritten, the data is lost.

If you want to acquire signal data into a single file, see .

Scope Trigger Configuration

You can configure xPC Target scopes to 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 a predefined trigger condition is met. The exact condition depends on the trigger mode that you select.

- `Freerun` — Acquires data as soon as the scope is started (default).
- `Software` — Acquires data in response to a user request, such as a call to a scope method `xpctarget.xpcsc.trigger` or the scope function `xPCScSoftwareTrigger`.
- `Signal` — Acquires data when a particular signal has crossed a preset level.
- `Scope` — Acquires data based on when another (triggering) scope starts.

You can use several properties to further refine when a scope starts to acquire data. For example, if you want the scope to be triggered when another signal crosses a certain value, use `Signal` trigger mode. Specify the following:

- The signal to trigger the scope.
- The trigger level that the signal must cross to trigger the scope to start acquiring data.
- Whether the scope is triggered on a rising signal, falling signal, or either one.

The *trigger point* is the sample at which the scope trigger condition is satisfied. For signal triggering, the trigger point is the sample at 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 pre- and post-triggering with the `NumPrePostSamples` scope property. See "Pre- and Post-Triggering of Scopes" on page 7-19.

Pre- and Post-Triggering of Scopes

By default, the scope starts acquiring data at the same time as the trigger event (the trigger point). In some cases, you want to observe the sequence of values that led to the trigger, so you start acquiring data a given number of samples before the trigger event (pre-triggering). In other cases, you want to observe the system settling down after the trigger event, so you delay acquiring data a given number of samples after the trigger event (post-triggering).

Use the `NumPrePostSamples` scope property to specify pre- and post-triggering. A negative value indicates pre-triggering and a positive value indicates post-triggering. For example, suppose `P` is the value of `NumPrePostSamples` for Scope 1 and `TP` is the trigger point, the sample where the trigger event occurs.

- `P = 0` — Scope 1 starts acquiring data immediately at trigger point `TP`.

Simple triggering (P = 0)

TP

First Sample
Acquired

End of
Acquisition

N
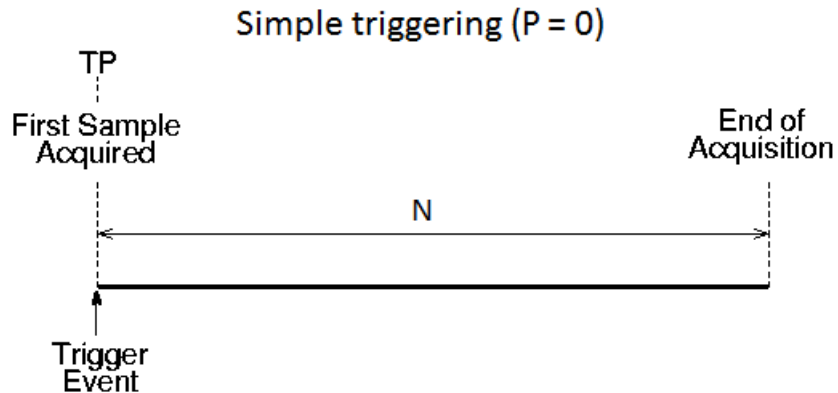
Trigger
Event

- `P < 0` — Scope 1 starts acquiring data |P| samples before trigger point TP.

Pre-triggering (P < 0)
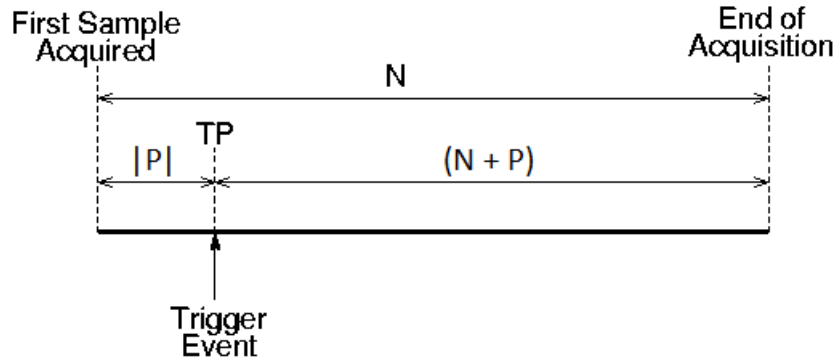
- `P > 0` — Scope 1 starts acquiring data P samples after trigger point TP.

Post-triggering (P > 0)
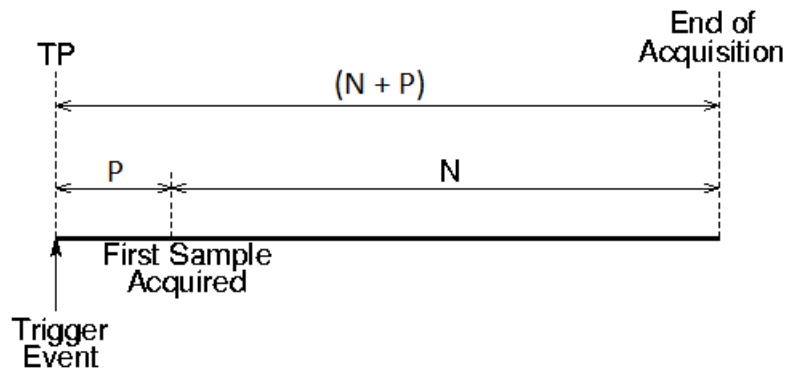
Trigger One Scope with Another Scope

When you have started two scopes that you want to keep synchronized, you can trigger one scope with another to acquire data. Set up the first scope with the trigger of your choice, and then trigger the second scope from the first.

In the following setup, Scope 1 triggers Scope 2.

1 Two scope objects are configured as a vector with the command:

```
sc = tg.addscope('host', [1 2]);
```

2 For Scope 1, set the following values:

```
sc(1).ScopeId = 1
sc(1).NumSamples = N1
sc(1).NumPrePostSamples = P1
```

3 For Scope 2, set the following values:

```
sc(2).ScopeId = 2
sc(2).NumSamples = N2
sc(2).TriggerMode = 'Scope'
sc(2).TriggerScope = 1
sc(2).NumPrePostSamples = P2
```

Because Scope 2 is triggered by Scope 1, the trigger point TP is the same for both scopes. However, different samples can be acquired by Scopes 1 and 2.

Scope-Triggered Data Acquisition

Some representative relationships between data acquisitions by Scope 1 and Scope 2 are shown in the following figures. P1 and P2 are the values of NumPrePostSamples for Scopes 1 and 2. TP is the trigger point, the sample where a trigger event occurs, for both Scopes 1 and 2. Scope 2 begins acquiring data as described.

- • P1 = 0 and P2 = 0 — Scopes 1 and 2 start acquiring data immediately at trigger point TP.
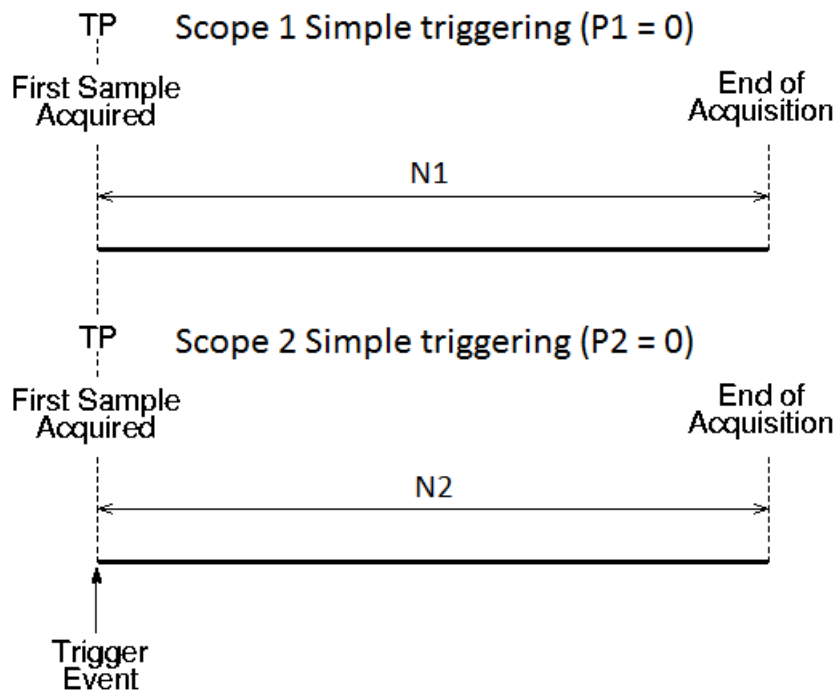
- P1 < 0 and P2 > 0 — Scope 1 starts acquiring data |P1| samples before trigger point TP. Scope 2 starts acquiring data P2 samples after trigger point TP.

Scope 1 Pre-triggering (P1 < 0)

First Sample Acquired

End of Acquisition

N1

TP

|P1|

(N1 + P1)

Scope 2 Post-triggering (P2 > 0)

TP

End of Acquisition

(N2 + P2)

P2

N2

First Sample Acquired

Trigger Event
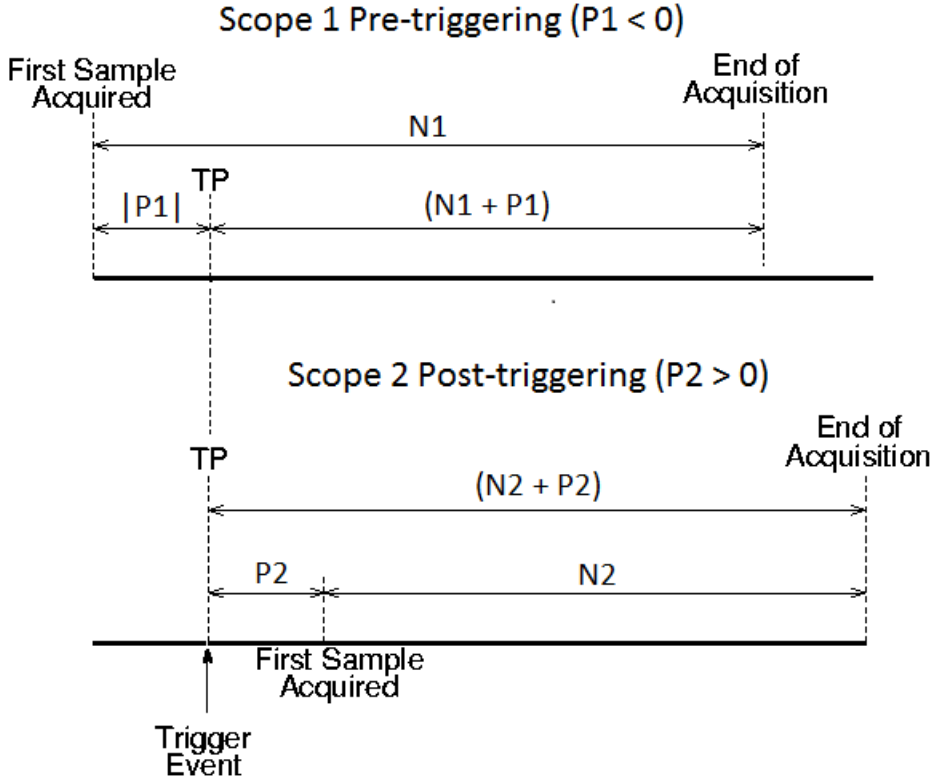
- `P1 > 0` and `P2 < 0`— Scope 1 starts acquiring data `P1` samples after trigger point TP. Scope 2 starts acquiring data `|P2|` samples before trigger point TP.
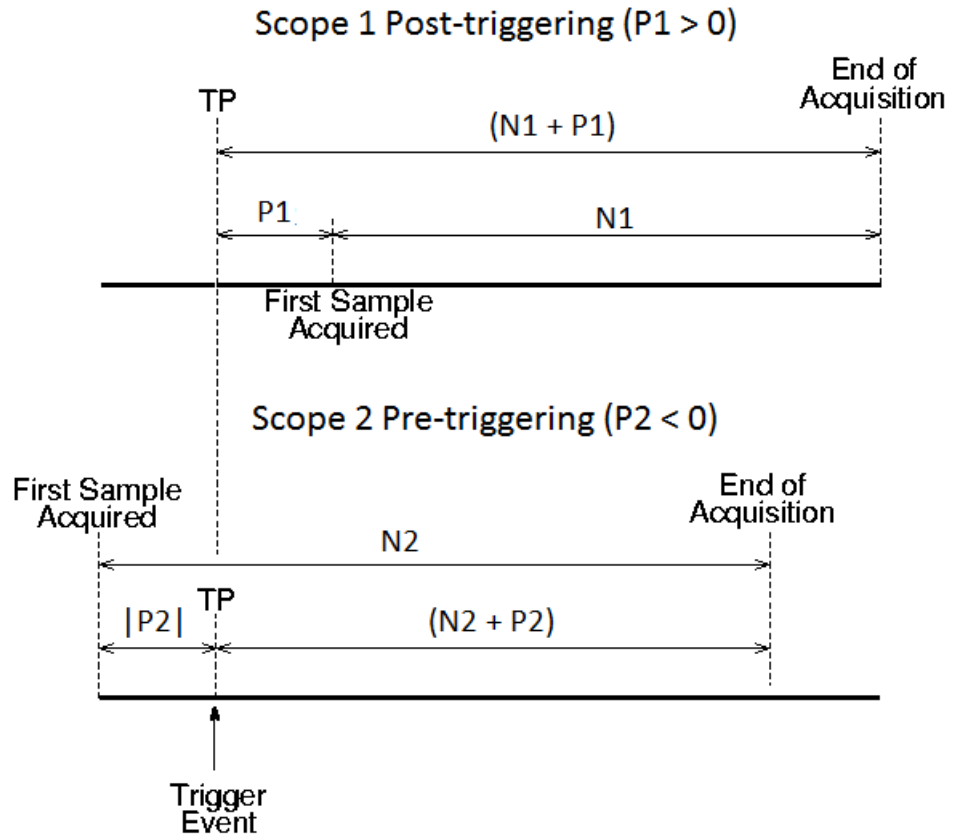
Trigger Sample Setting

For additional flexibility in scope triggering, you can use the Scope 2 trigger sample setting.

```
sc(2).TriggerSample = range 0 to (N + P1 - 1)
```

- `sc(2).TriggerSample = 0` (default) — Scope 2 triggers when Scope 1 triggers. Trigger point TP is the same sample for both scopes.
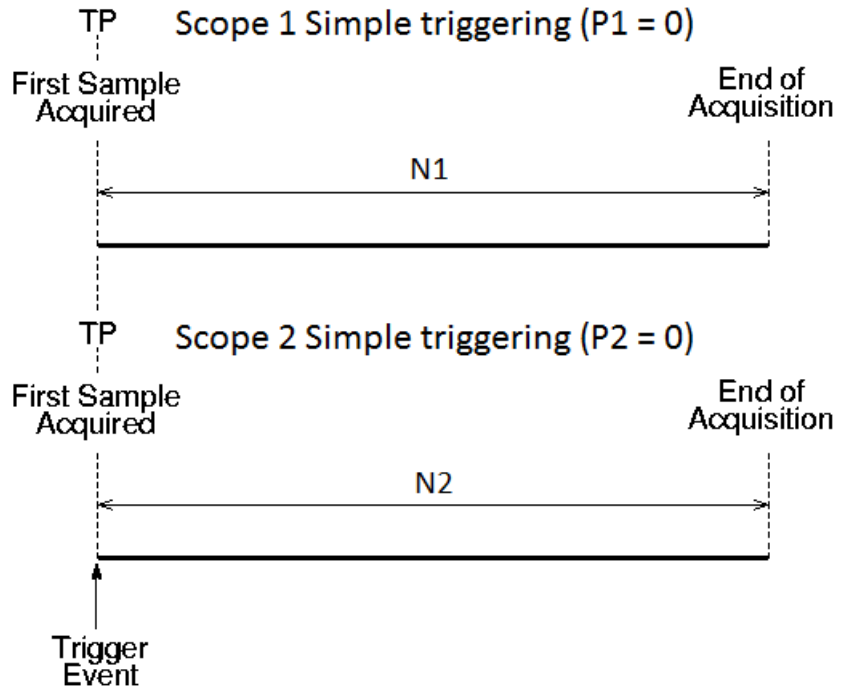
- `sc(2).TriggerSample = ts > 0` — Scope 2 triggers `ts` samples after Scope 1 is triggered. Trigger point `TP2` for Scope 2 is `ts` samples after `TP1` for Scope 1.

Scope 2 Delayed triggering (ts > 0)

TP1

First Sample
Acquired

N1

End of
Acquisition

TP2

First Sample
Acquired

ts

N2

End of
Acquisition
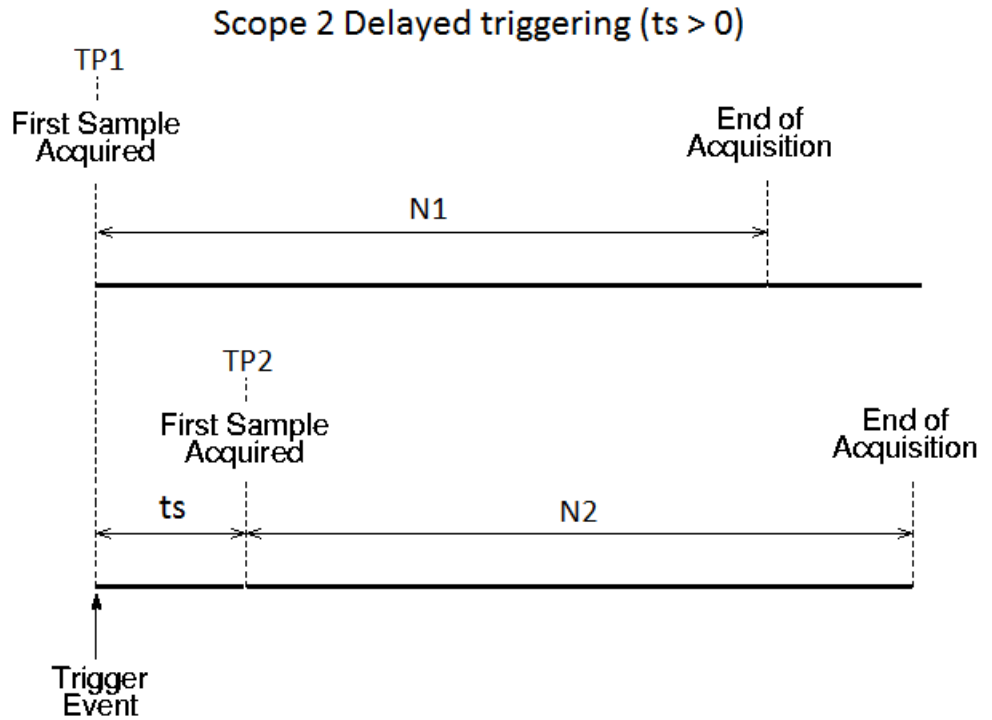
Trigger
Event

Setting `sc(2).TriggerSample` to a value `ts` larger than (`N + P - 1`) does not cause an error. It implies that Scope 2 cannot be triggered, because Scope 1 cannot acquire more than (`N + P - 1`) samples after `TP`.

- `sc(2).TriggerSample = -1` (special case) — Causes Scope 2 to start acquiring data from the sample after Scope 1 stops acquiring.

Scope 2 Special delayed triggering (ts = -1)
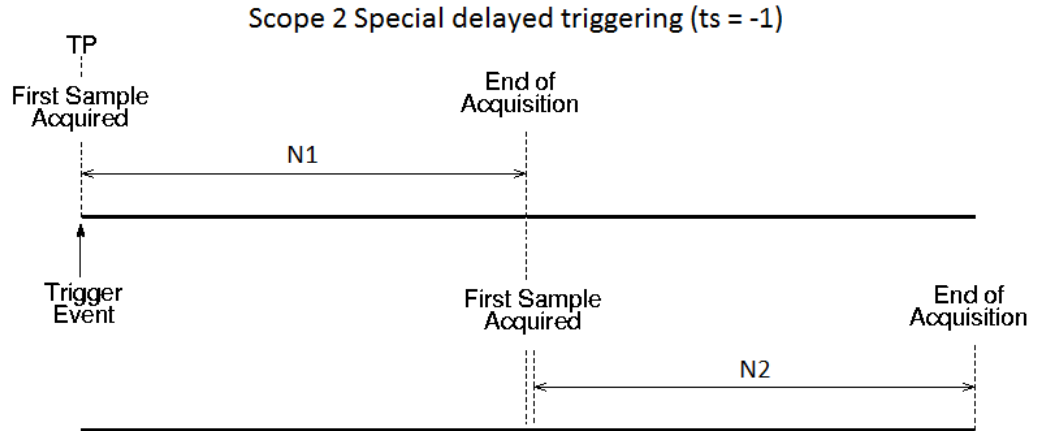
Acquire Gap-Free Data Using Two Scopes

With two scopes, you can acquire gap-free data. Gap-free data is data that two scopes acquire consecutively, without overlap. The first scope acquires data up to sample N, then stops. The second scope begins to acquire data at sample N+1.

In the following example, the TriggerMode property of Scope 1 is set to 'Software'. This setting allows Scope 1 to be triggered when it receives the MATLAB command sc1.trigger.
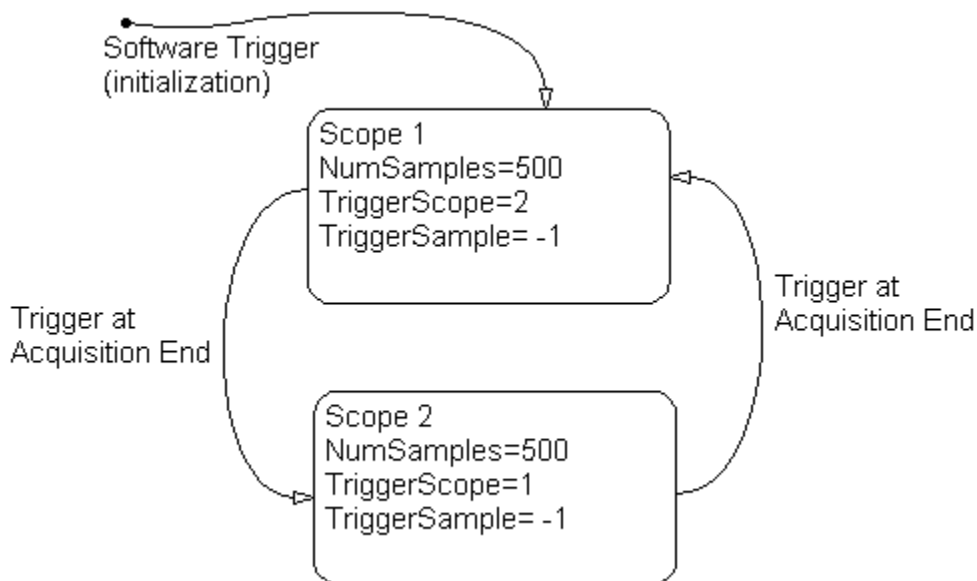
To programmatically acquire gap-free data with two scopes:

1 Build and download the Simulink model xpcosc to the target computer.

2 In the MATLAB Command Window, assign tg to the target computer and set the StopTime property to 1. For example:

```
tg=xpctarget.xpc
tg.StopTime = 1;
```

3 Add two host scopes to the target application. You can assign the two scopes to a vector, `sc`, so that you can work with both scopes with one command.

```
sc = tg.addscope('host', [1 2]);
```

4 Add the signals of interest (0 and 1) to both scopes.

```
addsignal(sc,[0 1]);
```

5 Set the `NumSamples` property for both scopes to `500` and the `TriggerSample` property for both scopes to `-1`. With this property setting, each scope triggers the next scope *at the end* of its 500 sample acquisition.

```
set(sc, 'NumSamples', 500, 'TriggerSample', -1)
```

6 Set the `TriggerMode` property for both scopes to `'Scope'`. Set the `TriggerScope` property such that each scope is triggered by the other.

```
set(sc, 'TriggerMode', 'Scope');
sc(1).TriggerScope = 2;
sc(2).TriggerScope = 1;
```

7 Set up storage for time, `t`, and signal, `data` acquisition.

```
t    = [];
data = zeros(0, 2);
```

8 Start both scopes and the model.

```
start(sc);
start(tg);
```

Both scopes receive exactly the same signals, 0 and 1.

9 Trigger scope 1 to start acquiring data.

```
scNum = 1;
sc(scNum).trigger;
```

Setting `scNum` to 1 indicates that scope 1 will acquire data first.

10 Start acquiring data using the two scopes to double buffer the data.

```
    while (1)
      % Wait until this scope has finished acquiring 500 samples
      % or the model stops (scope is interrupted).
      while ~(strcmp(sc(scNum).Status, 'Finished') || ...
              strcmp(sc(scNum).Status, 'Interrupted')), end
      % Stop buffering data when the model stops.
      if strcmp(tg.Status, 'stopped')
        break
      end
      % Save the data.
      t(   end + 1 : end + 500)    = sc(scNum).Time;
      data(end + 1 : end + 500, :) = sc(scNum).Data;
      % Restart this scope.
      start(sc(scNum));
      % Switch to the next scope.
    %Shortcut for if(scNum==1) scNum=2;else scNum=1,end
    scNum = 3 - scNum;
    end
```

11 When done, remove the scopes.

```
  % Remove the scopes we added.
  remscope(tg,[1 2]);
```

Following is a complete code listing for the preceding double-buffering data acquisition procedure. After you download the model (xpcosc) to the target computer, you can copy and paste this code into a MATLAB file and run it. The communication speed between the host and target computer must be fast enough to handle the number of samples and can acquire the full data set before the next acquisition cycles starts. In a similar way, you can use more than two scopes to implement a triple- or quadruple-buffering scheme.

```
% Assumes model xpcosc.mdl has been built and loaded on the target computer.
% Attach to the target computer and set StopTime to 1 sec.
tg = xpctarget.xpc;
tg.StopTime = 1;
% Add two host scopes.
sc = tg.addscope('host', [1 2]);
% [0 1] are the signals of interest.  Add to both scopes.
addsignal(sc,[0 1]);
% Each scope triggers next scope at end of a 500 sample acquisition.
```

```
set(sc, 'NumSamples', 500, 'TriggerSample', -1);
set(sc, 'TriggerMode', 'Scope');
sc(1).TriggerScope = 2;
sc(2).TriggerScope = 1;
% Initialize time and data log.
t    = [];
data = zeros(0, 2);
% Start the scopes and the model.
start(sc);
start(tg);
% Start things off by triggering scope 1.
scNum = 1;
sc(scNum).trigger;
% Use the two scopes as a double buffer to log the data.
while (1)
  % Wait until this scope has finished acquiring 500 samples
  % or the model stops (scope is interrupted).
  while ~(strcmp(sc(scNum).Status, 'Finished') || ...
          strcmp(sc(scNum).Status, 'Interrupted')), end
  % Stop buffering data when the model stops.
  if strcmp(tg.Status, 'stopped')
    break
  end
  % Save the data.
  t(   end + 1 : end + 500)    = sc(scNum).Time;
  data(end + 1 : end + 500, :) = sc(scNum).Data;
  % Restart this scope.
  start(sc(scNum));
  % Switch to the next scope.
  scNum = 3 - scNum;
end
% Remove the scopes we added.
remscope(tg,[1 2]);
% Plot the data.
plot(t,data); grid on; legend('Signal 0','Signal 1');
```

8

Logging Signal Data with FTP and File System Objects

File Systems

xPC Target file scopes create files on the target computer. To work with these files from the host computer, you need to work with the `xpctarget.ftp` and `xpctarget.fs` objects. The `xpctarget.ftp` object allows you to perform basic file transfer operations on the target computer file system. The `xpctarget.fs` object allows you to perform file system-like operations on the target computer file system.

You cannot direct the scope to write the data to a file on the xPC Target host computer. Once the software has written the signal data file to the target computer, you can access the contents of the file for plotting or other inspection from the host computer. The software can write data files to

- The C:\ or D:\ drive of the target computer. This can be a serial ATA (SATA) or parallel ATA (PATA)/Integrated Device Electronics (IDE) drive. The xPC Target software supports file systems of type FAT-12, FAT-16, or FAT-32. Verify that the hard drive is not cable-selected and that the BIOS can detect it. The type of file system (FAT-12, FAT-16, or FAT-32) limits the maximum size of the file. The target computer file system uses the 8.3 file name convention. This means that a target computer file name cannot exceed eight characters. Its file extension cannot exceed 3 characters.

  If you have a target computer with multiple partitions on a hard drive, the xPC Target software file scope can access those partitions if they are formatted with FAT-12, FAT-16, or FAT-32. It will ignore unsupported file systems.

- A 3.5-inch disk drive.

- Disks connected to a secondary IDE controller. The software supports up to four drives through the second IDE controller. By default, it works with drives configured as the primary master. If you want to use a secondary IDE controller, you must configure the xPC Target software for it (see "Converting xPC Target File Format Content to Double Precision Data" on page 8-14). The software searches for another drive in the first four ports of the target computer.

The largest single file that you can create is 4 GB.

Note that writing data files to 3.5-inch disk drives is considerably slower than writing to hard drives.

You can access signal data files, or other target computer system files, in one of the following ways:

- If you are running the target computer as a standalone system, you can access that file by rebooting the target computer under an operating system such as DOS and accessing the file through the operating system utilities.

- If you are running the target computer in conjunction with a host computer, you can access the target computer file from the host computer by representing that file as an `xpctarget.ftp` object. Through the MATLAB interface, use `xpctarget.ftp` methods on that FTP object. The `xpctarget.ftp` object methods are file transfer operations such as `get` and `put`.

- If you are running the target computer in conjunction with a host computer, you can access the target computer file from the host computer by representing the target computer file system as an `xpctarget.fs` object. Through the MATLAB interface, use the `xpctarget.fs` methods on the file system and perform file system-like methods such as `fopen` and `fread` on the signal data file. These methods work like the MATLAB file I/O methods. The `xpctarget.fs` methods also include file system utilities that allow you to collect target computer file system information for the disk and disk buffers.

This topic describes procedures on how to use the `xpctarget.ftp` and `xpctarget.fs` methods for common operations.

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

For an example of how to perform data logging with file scopes, see Data Logging With a File Scope.

FTP and File System Objects

The xPC Target software uses two objects, `xpctarget.ftp` and `xpctarget.fs` (file system), to work with files on a target computer. You use the `xpctarget.ftp` object to perform file transfer operations between the host and target computer. You use the `xpctarget.fs` object to access the target computer file system. For example, you can use an `xpctarget.fs` object to open, read, and close a signal data file created by an xPC Target file scope.

Note This feature provides FTP-like commands, such as `get` and `put`. However, it is not a standard FTP implementation. For example, the software does not support the use of a standard FTP client.

To create an `xpctarget.ftp` object, use the FTP object constructor function `xpctarget.ftp`. In the MATLAB Command Window, type

```
f = xpctarget.ftp
```

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

To create an `xpctarget.fs` object, use the FTP object constructor function `xpctarget.fs`. In the MATLAB window, type

```
f = xpctarget.fs
```

Both `xpctarget.ftp` and `xpctarget.fs` belong to the `xpctarget.fsbase` object. This object encompasses the methods common to `xpctarget.ftp` and `xpctarget.fs`. You can call the `xpctarget.fsbase` methods for both `xpctarget.ftp` and `xpctarget.fs` objects. The xPC Target software creates the `xpctarget.fsbase` object when you create either an `xpctarget.ftp` or `xpctarget.fs` object. You enter `xpctarget.fsbase` object methods in the MATLAB Command Window on the host computer or use MATLAB code scripts.

Using xpctarget.ftp Objects

In this section...
"Overview" on page 8-5
"Accessing Files on a Specific Target Computer" on page 8-6
"Listing the Contents of the Target Computer Folder" on page 8-7
"Retrieving a File from the Target Computer to the Host Computer" on page 8-8
"Copying a File from the Host Computer to the Target Computer" on page 8-8

Overview

The xpctarget.ftp object enables you to work with files on the target computer, including the data file that you generate from an xPC Target scope object of type file. You enter target object methods in the MATLAB window on the host computer or use scripts. The xpctarget.ftp object has methods that allow you to use

- xpctarget.fsbase.cd to change folders

- xpctarget.fsbase.dir to list the contents of the current folder

- xpctarget.fsbase.mkdir to make a folder

- xpctarget.fsbase.pwd to get the current working folder path

- xpctarget.fsbase.rmdir to remove a folder

- xpctarget.ftp.get (ftp) to retrieve a file from the target computer to the host computer

- xpctarget.ftp.put to place a file from the host computer to the target computer

The procedures in this section assume that the target computer has a signal data file created by an xPC Target file scope. This file has the pathname C:\data.dat. See "Configure Host Scope (xPC) Blocks" on page 5-27 for additional details.

The xPC Target software also provides methods that allow you to perform file system-type operations, such as opening and reading files. For a complete list of these methods, see "Using xpctarget.fs Objects" on page 8-10.

Accessing Files on a Specific Target Computer

You can access specific target computer files from the host computer for the `xpctarget.ftp` object.

Use the `xpctarget.ftp` creator function. If your system has multiple targets, you can access specific target computer files from the host computer for the `xpctarget.ftp` object.

For example, to list the name of the current folder of a target computer through a TCP/IP connection,

1 In the MATLAB Command Window, type a command like the following to assign the `xpctarget.ftp` object to a variable.

```
f=xpctarget.ftp('TCPIP','192.168.0.10','22222');
```

2 Type

```
f.pwd;
```

Alternatively, you can use the `xpctarget.xpc` constructor to first construct a target object, then use that target object as an argument to `xpctarget.ftp`.

1 In the MATLAB window, type a command like the following to assign the `xpctarget.xpc` object to a variable.

```
tg1=xpctarget.xpc('TCPIP','192.168.0.10','22222');
```

2 Type the following command to assign the `xpctarget.ftp` object to the `tg1` target object variable.

```
f=xpctarget.ftp(tg1);
```

Alternatively, if you want to work with the files of the default target computer, you can use the `xpctarget.ftp` constructor without arguments.

In the MATLAB window, type a command like the following to assign the xpctarget.ftp object to a variable.

```
f=xpctarget.ftp;
```

The xPC Target software assigns the f variable to the default target computer.

Listing the Contents of the Target Computer Folder

You can list the contents of the target computer folder by using xPC Target methods on the host computer for the xpctarget.ftp object. Use the method syntax to run an xpctarget.ftp object method:

```
method_name(ftp_object)
```

Note You must use the dir(f) syntax to list the contents of the folder. To get the results in an M-by-1 structure, use a syntax like y=dir(f). See the xpctarget.fsbase.dir method reference for further details.

For example, to list the contents of the C:\ drive,

1 In the MATLAB window, type the following to assign the xpctarget.ftp object to a variable:

```
f=xpctarget.ftp;
```

2 Type

```
f.pwd
```

This gets the current folder. You get a result like the following:

```
ans =
C:\
```

3 Type the following to list the contents of this folder:

```
dir(f)
```

Retrieving a File from the Target Computer to the Host Computer

You can retrieve a copy of a data file from the target computer by using xPC Target methods on the host computer for the `xpctarget.ftp` object.

Use the method syntax to run an `xpctarget.ftp` object method. The syntax `method_name(ftp_object, argument_list)` can be replaced with

```
ftp_object.method_name(argument_list)
```

For example, to retrieve a file named `data.dat` from the target computer C:\ drive (default),

1 If you have not already done so, in the MATLAB window, type the following to assign the `xpctarget.ftp` object to a variable.

```
f=xpctarget.ftp;
```

2 Type

```
f.get('data.dat');
```

This retrieves the file and saves that file to the variable `data`. This content is in the xPC Target file format.

Copying a File from the Host Computer to the Target Computer

You can place a copy of a file from the host computer by using xPC Target methods on the host computer for the `xpctarget.ftp` object.

Use the method syntax to run an `xpctarget.ftp` object method. The syntax `method_name(ftp_object, argument_list)` can be replaced with

```
ftp_object.method_name(argument_list)
```

For example, to copy a file named `data2.dat` from the host computer to the target computer C:\ drive (default),

1 If you have not already done so, in the MATLAB window, type the following to assign the `xpctarget.ftp` object to a variable.

```
f=xpctarget.ftp;
```

2 Type the following to save that file to the variable `data`.

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

Using xpctarget.fs Objects

Overview

The `fs` object enables you to work with the target computer file system from the host computer. You enter target object methods in the MATLAB window on the host computer or use scripts. The `fs` object has methods that allow you to use

- `xpctarget.fsbase.cd` to change folders
- `xpctarget.fsbase.dir` to list the contents of the current folder
- `xpctarget.fsbase.mkdir` to make a folder
- `xpctarget.fsbase.pwd` to get the current working folder path
- `xpctarget.fsbase.rmdir` to remove a folder
- `xpctarget.fs.diskinfo` to get information about the specified disk
- `xpctarget.fs.fclose` to close a file (similar to MATLAB `fclose`)
- `xpctarget.fs.fileinfo` to get information about a particular file
- `xpctarget.fs.filetable` to get information about files in the file system
- `xpctarget.fs.fopen` to open a file (similar to MATLAB `fopen`)
- `xpctarget.fs.fread` to read a file (similar to MATLAB `fread`)

- `xpctarget.fs.fwrite` to write a file (similar to MATLAB `fwrite`)

- `xpctarget.fs.getfilesize` to get the size of a file in bytes

- `xpctarget.fs.removefile` to remove a file from the target computer

Useful global utility:

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

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

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

Accessing File Systems from a Specific Target Computer

You can access specific target computer files from the host computer for the `xpctarget.fs` object.

Use the `xpctarget.fs` creator function. If your system has multiple targets, you can access specific target computer files from the host computer for the `xpctarget.fs` object.

For example, to list the name of the current folder of a target computer through a TCP/IP connection,

1 In the MATLAB window, type a command like the following to assign the `xpctarget.fs` object to a variable.

```
fsys=xpctarget.fs('TCPIP','192.168.0.10','22222');
```

2 Type

```
fsys.dir;
```

Alternatively, you can use the `xpctarget.xpc` constructor to first construct a target object, then use that target object as an argument to `xpctarget.fs`.

1 In the MATLAB window, type a command like the following to assign the `xpctarget.xpc` object to a variable.

```
tg1=xpctarget.xpc('TCPIP','192.168.0.10','22222');
```

2 Type the following command to assign the `xpctarget.fs` object to the `tg1` target object variable.

```
fs=xpctarget.fs(tg1);
```

Alternatively, if you want to work with the file system of the default target computer, you can use the `xpctarget.fs` constructor without arguments.

1 In the MATLAB window, type a command like the following to assign the `xpctarget.fs` object to a variable.

```
fsys=xpctarget.fs;
```

The xPC Target software assigns the `fsys` variable to the default target computer.

2 Type

```
fsys.dir;
```

Retrieving the Contents of a File from the Target Computer to the Host Computer

You can retrieve the contents of a data file from the target computer by using xPC Target methods on the host computer for the `xpctarget.fs` object. This is an alternate method to "Configure File Scopes Using MATLAB Language" on page 5-106.

Use the method syntax to run an `xpctarget.fs` object method. The syntax `method_name(fs_object, argument_list)` can be replaced with

```
fs_object.method_name(argument_list)
```

For example, to retrieve the contents of a file named `data.dat` from the target computer C:\ drive (default),

1 If you have not already done so, in the MATLAB window, type the following to assign the `xpctarget.fs` object to a variable.

```
fsys=xpctarget.fs;
```

2 Type

```
h=fsys.fopen('data.dat');
```

or

```
h=fopen(fsys,'data.dat');
```

This opens the file `data.dat` for reading and assigns the file identifier to `h`.

3 Type

```
data2=fsys.fread(h);
```

or

```
data2=fread(fsys,h);
```

This reads the file `data.dat` and stores the contents of the file to `data2`. This content is in the xPC Target file format.

4 Type

```
fsys.fclose(h);
```

This closes the file `data.dat`.

Before you can view or plot the contents of this file, you must convert the contents. See "Converting xPC Target File Format Content to Double Precision Data" on page 8-14.

Converting xPC Target File Format Content to Double Precision Data

The xPC Target software provides the script `readxpcfile.m` to convert xPC Target file format content (in bytes) to double precision data representing the signals and timestamps. The `readxpcfile.m` script takes in data from a file in xPC Target format. The data must be a vector of bytes (`uint8`). To convert the data to `uint8`, use a command like the following:

```
data2 = uint8(data2');
```

This section assumes that you have a variable, `data2`, that contains data in the xPC Target file format (see "Retrieving the Contents of a File from the Target Computer to the Host Computer" on page 8-12):

1 In the MATLAB window, change folder to the folder that contains the xPC Target format file.

2 Type

```
new_data2=readxpcfile(data2);
```

The `readxpcfile` script converts the format of `data2` from the xPC Target file format to an array of bytes. It also creates a structure for that file in `new_data2`, of which one of the elements is an array of doubles, `data`. The `data` member is also appended with a time stamp vector. The data is returned as doubles, which represent the real-world values of the original Simulink signals at the specified times during target execution.

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

If you are using the xPC Target software in StandAlone mode, you can extract the data from the data file if you know the number of signals in the scope and file header size. If you know these numbers, you can extract the data. Note the following:

- First determine the file header size. To obtain the file header size, ignore the first eight bytes of the file. The next four bytes store the header size as an unsigned integer.

- After the header size number of bytes, the file stores the signals sequentially as doubles. For example, assume the scope has three signals, x, y, and z. Assume that x[0] is the value of x at sample 0, x[1] is the value at sample 1, and so forth, and t[0], t[1] are the simulation time values at samples 0, 1, and so forth, respectively. The file saves the data using the following pattern:

  ```
  x[0] y[0] z[0] t[0] x[1] y[1] z[1] t[1] x[2] y[2] z[2] t[2]...
  x[N] y[N] z[N] t[N]
  ```

  N is the number of samples acquired. The file saves x, y, z, and t as doubles at 8 bytes each.

Removing a File from the Target Computer

You can remove a file from the target computer by using xPC Target methods on the host computer for the xpctarget.ftp object. If you have not already done so, close this file first with fclose.

Use the method syntax to run an xpctarget.fs object method. The syntax method_name(fs_object, argument_list) can be replaced with

```
fs_object.method_name(argument_list)
```

For example, to remove a file named data2.dat from the target computer C:\ drive (default),

1 If you have not already done so, in the MATLAB window, type the following to assign the xpctarget.fs object to a variable.

```
fsys=xpctarget.fs;
```

2 Type the following to remove the specified file from the target computer.

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

or

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

Getting a List of Open Files on the Target Computer

You can get a list of open files on the target computer file system from the host computer by using xPC Target methods on the host computer for the `xpctarget.fs` object. Do this to identify files you can close. The target computer file system limits the number of open files you can have to eight.

Use the method syntax to run an `xpctarget.fs` object method. The syntax `method_name(fs_object, argument_list)` can be replaced with

```
fs_object.method_name(argument_list)
```

For example, to get a list of open files for the file system object `fsys`,

1 If you have not already done so, in the MATLAB window, type the following to assign the `xpctarget.fs` object to a variable.

```
fsys=xpctarget.fs;
```

2 Type

```
fsys.filetable
```

If the file system has open files, a list like the following is displayed:

```
ans =
Index    Handle  Flags      FilePos  Name
------------------------------------------
    0  00060000  R__           8512  C:\DATA.DAT
    1  00080001  R__              0  C:\DATA1.DAT
    2  000A0002  R__           8512  C:\DATA2.DAT
    3  000C0003  R__           8512  C:\DATA3.DAT
    4  001E0001  R__              0  C:\DATA4.DA
```

3 The table returns the open file handles in hexadecimal. To convert a handle to one that other `xpctarget.fs` methods, such as `fclose`, can use, use the `hex2dec` function. For example,

```
h1 = hex2dec('001E0001'))
h1 =
1966081
```

4 To close that file, use the `xpctarget.fs` `fclose` method. For example,

```
fsys.fclose(h1);
```

Getting Information about a File on the Target Computer

You can display information for a file on the target computer file system from the host computer by using xPC Target methods on the host computer for the `xpctarget.fs` object.

Use the method syntax to run an `xpctarget.fs` object method. The syntax `method_name(fs_object, argument_list)` can be replaced with

```
fs_object.method_name(argument_list)
```

For example, to display the information for the file identifier `fid1`,

1 If you have not already done so, in the MATLAB window, type the following to assign the `xpctarget.fs` object to a variable.

```
fsys=xpctarget.fs;
```

2 Type

```
fid1=fsys.fopen('data.dat');
```

This opens the file `data.dat` for reading and assigns the file identifier to `fid1`.

3 Type

```
fsys.fileinfo(fid1);
```

This returns disk information like the following for the C:\ drive file system.

```
ans =
                FilePos: 0
          AllocatedSize: 12288
          ClusterChains: 1
    VolumeSerialNumber: 1.0450e+009
               FullName: 'C:\DATA.DAT'
```

Getting Information about a Disk on the Target Computer

You can display information for a disk on the target computer file system from the host computer by using xPC Target methods on the host computer for the `xpctarget.fs` object.

Use the method syntax to run an `xpctarget.fs` object method. The syntax `method_name(fs_object, argument_list)` can be replaced with

```
fs_object.method_name(argument_list)
```

For example, to display the disk information for the C:\ drive,

1 If you have not already done so, in the MATLAB window, type the following to assign the `xpctarget.fs` object to a variable.

```
fsys=xpctarget.fs;
```

2 Type

```
fsys.diskinfo('C:\');
```

This returns disk information like the following for the C:\ drive file system.

```
ans =

                 Label: 'SYSTEM '
           DriveLetter: 'C'
              Reserved: ''
          SerialNumber: 1.0294e+009
     FirstPhysicalSector: 63
               FATType: 32
              FATCount: 2
          MaxDirEntries: 0
        BytesPerSector: 512
      SectorsPerCluster: 4
          TotalClusters: 2040293
            BadClusters: 0
           FreeClusters: 1007937
                 Files: 19968
            FileChains: 22480
```

```
        FreeChains: 1300
  LargestFreeChain: 64349
```

9

Execution Using Graphical User Interface Models

You can use the Simulink interface to create a custom graphical user interface (GUI) for your xPC Target application. To do this, create an user interface model with the Simulink interface and add-on products like Simulink 3D Animation™ or Altia® Design (a third-party product).

xPC Target Interface Blocks to Simulink Models

Simulink User Interface Model

A user interface model is a Simulink model containing Simulink blocks from add-on products and interface blocks from the xPC Target block library. This user interface model can connect to a custom graphical interface using Simulink 3D Animation or Altia products. The user interface model runs on the host computer and communicates with your target application running on the target computer using To xPC Target and From xPC Target blocks.

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

Simulink 3D Animation — The Simulink 3D Animation product enables you to display a Simulink user interface model in 3-D. It provides Simulink blocks that communicate with xPC Target interface blocks. These blocks then communicate to a graphical interface. This graphical interface is a Virtual Reality Modeling Language (VRML) world displayed with a Web browser using a VRML plug-in.

Altia Design — Altia also provides Simulink blocks that communicate with xPC Target interface blocks. These blocks then communicate with Altia's graphical interface or with a Web browser using the Altia ProtoPlay plug-in.
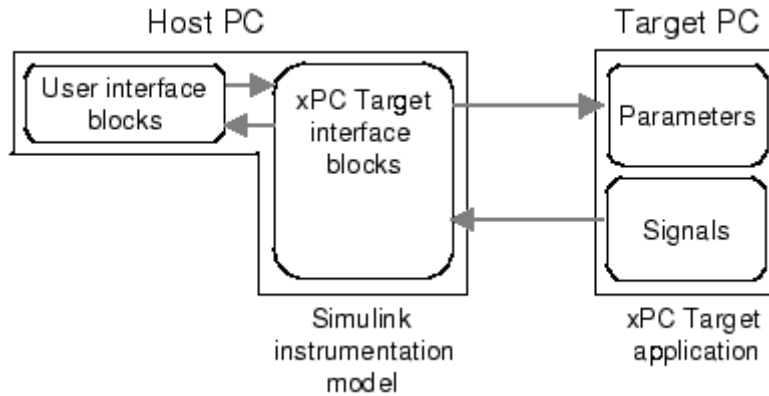
Creating a Custom Graphical Interface

The xPC Target block library provides Simulink interface blocks to connect graphical interface elements to your target application. The steps for creating your own custom user interface are listed below:

1 In the Simulink target application model, decide which block parameters and block signals you want to have access to through graphical interface control devices and graphical interface display devices.

2 Tag the block parameters in the Simulink model that you want to be connected to a control device. See "Marking Block Parameters" on page 9-6.

3 Tag the signals in Simulink model that you want to be connected to a display device. See "Marking Block Signals" on page 9-8.

4 In the MATLAB interface, 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.

5 To the user interface template model, add Simulink interface blocks from add-on products (Simulink 3D Animation, Altia Design).

- You can connect Altia blocks to the xPC Target To PC Target interface blocks. To xPC Target blocks on the left should be connected to control devices.

- You can connect Altia and Simulink 3D Animation blocks to the xPC Target From PC Target interface blocks. From xPC Target blocks on the right should be connected to the display devices.

You can position these blocks to your liking.

6 Start both the xPC Target application and the Simulink user interface model that represents the xPC Target application.

To xPC Target Block

This block behaves as a sink and usually receives its input data from a control device. The purpose of this block is to write a new value to a specific parameter on the target application.
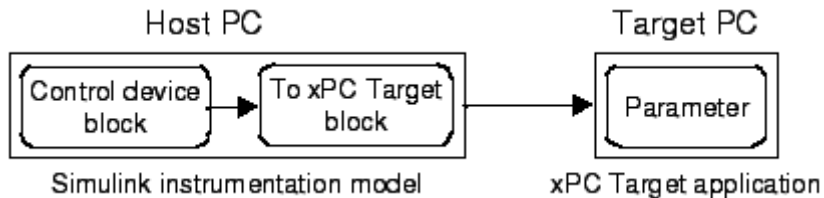
This block is implemented as a MATLAB S-function. The block is optimized so that it only changes a parameter on the target application when the input value differs from the value that existed at the last time step. This block uses the parameter downloading feature of the xPC Target command-line interface. This block is available from the xpclib/Misc block sublibrary. See To xPC Target for further configuration details.

Note The use of To xPC Target blocks requires a connection between the host and target computer. Operations such as opening a model that contains these blocks or copying these blocks within or between models will take significantly longer than normal without a connection between the host and target computers.

From xPC Target Block

This block behaves like a source and its output is usually connected to the input of a display device.
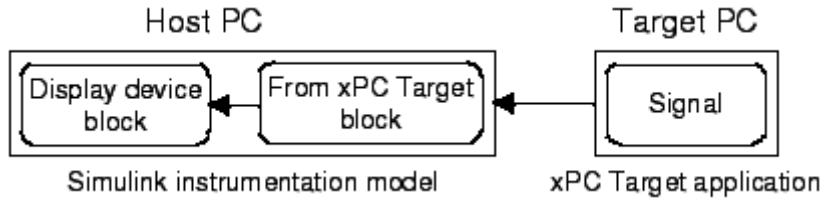
Because only one numerical value per signal is uploaded during a time step, the number of samples of a scope object is set to 1. The block uses the capability of the xPC Target command-line interface and is implemented as a MATLAB S-function. This block is available from the xpclib/Misc sublibrary. See From xPC Target for further configuration details.

Note The use of From xPC Target blocks requires a connection between the host and target computers. Operations such as opening a model that contains these blocks or copying these blocks within or between models will take significantly longer than normal without a connection between the host and target computers.

Creating a Target Application Model

A target application model is a Simulink model that describes your physical system, a controller, and its behavior. You use this model to create a real-time target application, and you use this model to select the parameters and signals you want to connect to a custom graphical interface.

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

See "Marking Block Parameters" on page 9-6 and "Marking Block Signals" on page 9-8 for descriptions of how to mark block properties and block signals.

Marking Block Parameters

Tagging parameters in your Simulink model allows the function xpcsliface to create To xPC Target interface blocks. These interface blocks contain the parameters you connect to control devices in your user interface model.

After you create a Simulink model, you can mark the block parameters. This procedure uses the model xpctank as an example.

Tip The xpctank model blocks and signals may contain placeholder tags illustrating the syntax. As you create your own copy of the model using these procedures, replace these tags with your new tags or add the new tags using the multiple label syntax.

---

1 Open a Simulink model. For example, in the MATLAB Command Window, type

   xpctank

2 Point to a Simulink block, and then right-click.

3 From the menu, click **Properties**.

   A Block Properties dialog box opens.

4 In the **Description** box, delete the existing tag and enter a tag to the parameters for this block.

   For example, the SetPoint block is a constant with a single parameter that selects the level of water in the tank. Enter the tag:

   xPCTag(1)=water_level;

   The tag has the following syntax

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

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

   xPCTag=label;

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

label_n -- Name for a block parameter that will be connected to a To xPC Target block in the user interface model. Separate the labels with a space, not a comma.

label_1...label_n must consist of the same identifiers as those used by C/C++ to name functions, variables, and so forth. Do not use names like -foo.

5 Repeat steps 1 through 3 for the remaining parameters you want to tag.

For example, for the Controller block, enter the tag:

```
xPCTag(1,2,3)=upper_water_level lower_water_level
    pump_flowrate;
```

For the PumpSwitch and ValveSwitch blocks, enter the following tags respectively:

```
xPCTag(2)=pump_switch;
xPCTag(1)=drain_valve;
```

To create the To xPC blocks in an user interface model for a block with four properties, use the following syntax:

```
xPCTag(1,2,3,4)=label_1label_2label_3label_4;
```

To create the To xPC blocks for the second and fourth properties in a block with at least four properties, use the following syntax:

```
xPCTag(2,4)=label_1 label_2;
```

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

```
xpctank1
```

You 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 9-8 and "Creating a Custom Graphical Interface" on page 9-3.

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 in your user interface model.

After you create a Simulink model, you can mark the block signals. This procedure uses the model xpctank1 (or xpctank) as an example. See "Creating a Target Application Model" on page 9-5.

Tip The xpctank model blocks and signals may contain placeholder tags illustrating the syntax. As you create your own copy of the model using these procedures, replace these tags with your new tags or add the new tags using the multiple label syntax.

Note that you cannot select signals on the output ports of virtual blocks, such as Subsystem and Mux blocks. Also, you cannot select signals on software-triggered signal output ports.

1 Open a Simulink model. For example, in the MATLAB Command Window, type:

xpctank

or

xpctank1

2 Point to a Simulink signal line, and then right-click.

3 From the menu, click **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:

```
xPCTag(1)=water_level;
```

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 Target 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 Target blocks for the second and fourth signals in a signal line with at least four signals, use the following syntax:

```
xPCTag(2,4)=label_1 label_2;
```

Note Only tag signals from nonvirtual blocks. Virtual blocks are only graphical aids (see "Virtual Blocks"). For example, if your model combines two signals into the inputs of a Mux block, do not tag the signal from the output of the Mux block. Instead, tag the source signal from the output of the originating nonvirtual block.

---

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

```
xpc_tank1
```

You next task is to mark block parameters if you have not already done so. See "Marking Block Parameters" on page 9-6. If you have already marked block signals, return to "Creating a Custom Graphical Interface" on page 9-3 for additional guidance on creating a user interface template model.

Execution Using the Target Computer Command Line

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

Target Computer Command-Line Interface

This interface is useful with standalone applications that are not connected to the host computer. You can type commands directly from a keyboard on the target computer. As you start to type at the keyboard, a command window appears on the target computer screen.

For a complete list of target computer commands, refer to "Target Computer Commands"

In this section...
"Using Target Application Methods on the Target Computer" on page 10-2
"Manipulating Target Object Properties from the Target Computer" on page 10-3
"Manipulating Scope Objects from the Target Computer" on page 10-4
"Manipulating Scope Object Properties from the Target Computer" on page 10-6
"Aliasing with Variable Commands on the Target Computer" on page 10-6

Using Target Application Methods on the Target Computer

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

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

1 On the target computer, press **C**.

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

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

```
start
```

3 To stop the application, type

```
stop
```

Once the command window is active, you do not have to reactivate it before typing the next command.

Manipulating Target Object Properties from the Target Computer

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

These commands create a temporary difference between the behavior of the target application and the properties of the target object. The next time you access the target object, the properties are updated from the target computer.

1 On the target computer keyboard, press **C**.

The target computer activates the command window.

2 Type a target command. For example, to change the frequency of the signal generator (parameter 1) in the model xpcosc, type

```
setpar 1=30
```

The command window displays a message to indicate that the new parameter has registered.

```
System: p[1] is set to 30.00000
```

3 Check the value of parameter 1. For example, type

```
p1
```

The command window displays a message to indicate that the new parameter has registered.

```
System: p[1] is set to 30.00000
```

4 Check the value of signal 0. For example, type

```
s0
```

The command window displays a message to indicate that the new parameter has registered.

```
System: S0 has value 5.1851
```

5 Change the stop time. For example, to set the stop time to 1000, type

```
stoptime = 1000
```

The parameter changes are made to the target application but not to the target object. When you type an xPC Target command in the MATLAB Command Window, the target computer returns the current properties of the target object.

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

To see the correlation between a parameter or signal index and its block, you can look at the `model_name_pt.c` or `model_name_bio.c` of the generated code for your target application.

Manipulating Scope Objects from the Target Computer

The xPC Target software uses a scope object to represent your target scope. This section shows some of the common tasks that you use with scope objects.

These commands create a temporary difference between the behavior of the target application and scope object. The next time you access the scope object, the data is updated from the target computer.

1 On the target computer keyboard, press **C**.

The target computer activates the command window.

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

```
addscope 2
```

The xPC Target software adds another scope monitor to the target computer screen. The command window displays a message to indicate that the new scope has registered.

```
Scope: 2, created, type is target SO
```

3 Type a scope command. For example, to add a signal (0) to the new scope, type

```
addsignal 2=0
```

The command window displays a message to indicate that the new parameter has registered.

```
Scope: 2, signal 0 added
```

You can add more signals to the scope.

4 Type a scope command. For example, to start the scope 2, type

```
startscope 2
```

The target scope 2 starts and displays the signals you added in the previous step.

Note If you add a target scope from the target computer, you need to start that scope manually. If a target scope is in the model, starting the target application starts that scope automatically.

Manipulating Scope Object Properties from the Target Computer

This section shows some of the common tasks that you use with target objects and their properties.

These commands create a temporary difference between the behavior of the target application and the properties of the target object. The next time you access the target object, the properties are updated from the target computer.

1 On the target computer keyboard, press **C**.

The target computer activates the command window.

2 Type a scope property command. For example, to change the number of samples (1000) to acquire in scope 2 of the model xpcosc, type

```
numsamples 2=1000
```

3 Type a scope property command. For example, to change the scope mode (numerical) of scope 2 of the model xpcosc, type

```
scopemode 2=numerical
```

The target scope 2 display changes to a numerical one.

Aliasing with Variable Commands on the Target Computer

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

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

1 On the target computer keyboard, type a variable command. For example, if you have a parameter that controls a motor, you could create the variables on and off by typing

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

The target computer command window is activated when you start to type, and a command line opens.

2 Type the variable name to run that command sequence. For example, to turn the motor on, type

```
on
```

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

Execution Using the Web Browser Interface

Web Browser Interface

Introduction

The xPC Target software has a Web server that allows you to interact with your target application through a Web browser. You can access the Web browser with either a TCP/IP or serial (RS-232) connection.

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

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

Connecting the Web Interface Through TCP/IP

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

The TCP/IP stack on the xPC Target kernel supports only one simultaneous connection, because its main objective is real-time applications. This

connection is shared between the MATLAB interface and the Web browser. You must close the other open connections to the target computer before you connect using the host computer Web browser. This also means that only one browser or the MATLAB interface is able to connect at one time.

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

Note Close the other connections to the target computer. For example, if you are currently connected to the target computer through xPC Target Explorer, right-click on that target computer icon and select **Disconnect** or click the Disconnect icon 🔵 on the toolbar.

1 In the MATLAB window, type

```
xpcwwwenable
```

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

2 Open a Web browser. In the address box, enter the IP address and port number you entered in the xPC Target Explorer window. For example, if the target computer IP address is 192.168.0.10 and the port is 22222, type

```
http://192.168.0.10:22222/
```

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

Connecting the Web Interface Through RS-232

If the host computer and target computer are connected with a serial cable instead of a network cable, you can still connect the target application on the target computer to a Web browser on the host computer. The xPC Target software includes a TCP/IP to RS-232 mapping application. This application runs on the host computer and writes whatever it receives from the RS-232

connection to a TCP/IP port, and it writes whatever is 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 computer, you must load a target application onto the target computer. The target application does not have to be running, but it must be loaded. Also, your Web browser must have JavaScript and StyleSheets turned on.

1 In the MATLAB window, type

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

The MATLAB interface is disconnected from the target computer, leaving the target computer ready to connect to another client. The TCP/IP stack of the xPC Target kernel supports only one simultaneous connection. If you do not use this command, the TCP/IP to RS-232 gateway might not be able to connect to the target computer.

2 Open a DOS command window, and enter the command to start the TCP/IP to RS-232 gateway. For example, if the target computer is connected to COM1 and you would like to use the TCP/IP port 22222, type the following:

```
c:\<MATLAB root>\toolbox\rtw\targets\xpc\xpc\bin\xpctcp2ser
-v -t 22222 -c 1
```

For a description of the xpctcp2ser command, see "Syntax for the xpctcp2ser Command" on page 11-5.

The TCP/IP to RS-232 gateway starts running, and the DOS command window displays the message

```
*-------------------------------------------------------------*
*            xPC Target TCP/IP to RS-232 gateway              *
*               Copyright 2000 The MathWorks                  *
*-------------------------------------------------------------*
Connecting COM to TCP port 22222
Waiting to connect
```

If you did not close the MATLAB to target application connection, xpxtcp2ser displays the message Could not initialize COM port.

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

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

The Web browser loads the xPC Target Web interface panes.

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

5 In the DOS command window, press **Ctrl+C**.

The TCP/IP to RS-232 Gateway stops running, and the DOS command window displays the message

```
interrupt received, shutting down
```

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

6 In the MATLAB Command Window, type

```
xpc
```

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

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

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

You must close the MATLAB interface and then restart it.

Syntax for the xpctcp2ser Command

The xpctcp2ser command starts the TCP/IP to RS-232 gateway. The syntax for this command is

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

The options are described in the following table.

Command-Line Option	Description
-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 computer that is connected to the target computer with a serial cable can connect to the selected port. For example, if you start the gateway on your host computer, with the default ports, you can type in the Web browser `http://localhost:2222`. However, if you try to connect to `http://Domainname.com:22222`, you will probably get a connection error.
-t tcpPort	Use TCP port `tcpPort`. Default t is 22222. For example, to connect to port 20010, type `-t 20010`.
-h	Print a help message.
-c comPort	Use COM port *comPort* ($1 <= comPort <= 4$). Default is 1. For example, to use COM2, type `-c 2`.

Using the Main Pane

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

After you connect a Web browser to the target computer, you can use the **Main** pane to control the target application:

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

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

2 Click the **Start Execution** button.

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

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

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

The new property values are downloaded to the target application. Note that the **SampleTime** box is visible only when the target application is stopped. You cannot change the sample time while a target application is running. (See "User Interaction" for limitations on changing sample times.)

5 Select scopes to view on the target computer. From the **ViewMode** list, select one or all of the scopes to view.

After entering values, the screen looks like this:

Note The **ViewMode** control is visible in the **xPC Target Properties** pane only if you add two or more scopes to the target computer.

Changing WWW Properties

The **WWW Properties** cell in the left frame contains fields that control 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` (show all signals), `1` (show only scalar signals), `2` (show scalar and vector signals less than or equal to 2 wide), or `n` (show signals with a width less than or equal to `n`).

Signals with a width greater than the value you enter are not displayed on the **Signals** pane.

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

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

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

This can also happen when you are trying to update a parameter or property at the same time that the pane is automatically refreshing.

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

Viewing Signals with a Web Browser

The **Signals** pane is a list of the signals in your model.

After you connect a Web browser to the target computer you can use the **Signals** pane to view signal data:

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

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

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

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

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

The **Screen Shot** pane is loaded and a copy of the current target computer screen is displayed. The screen shot uses the portable network graphics (PNG) file format.

Viewing Parameters with a Web Browser

The **Parameters** pane displays the tunable parameters in your model. Row and column indices for vector/matrix parameters are also shown.

After you connect a Web browser to the target computer, you can use the **Parameters** pane to change parameters in your target application while it is running in real time:

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

The **Parameter List** pane is loaded into the right frame.

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

If the parameter is a vector or matrix, click the **Edit** button to view the vector or matrix. You can edit the parameter in this pane.

2 In the **Value** box, enter a new parameter value, and then click the **Apply** button.

Changing Access Levels to the Web Browser

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

1 In the Simulink window, click **Simulation > Model Configuration Parameters**.

The Configuration Parameters dialog box for the model is displayed.

2 Click the **Code Generation** node.

The code generation pane opens.

3 In the **Target selection** section, access levels are set in the **System target file** box. For example, to set the access level to 1, enter

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

If you do not specify -axpcWWWAccessLevel, 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 the panes and functions.

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

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

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

Level 4 — Cannot edit existing scopes on the **Scopes** pane. Cannot add or remove signals on the **Scopes** pane. Cannot view the **Signals** pane and the **Parameters** pane, and cannot get scope data.

Troubleshooting

Refer to these guidelines, hints, and tips for questions
or issues you might have about your installation of the xPC
Target product. For more specific troubleshooting solutions,
go to the MathWorks® Support xPC Target Web site
(http://www.mathworks.com/support/search_results.html?q=product:"xPC+T
for specific troubleshooting solutions.

Troubleshooting Procedure

An xPC Target installation can sometimes fail. Causes include hardware failures, changes in underlying system software, and procedural errors. Follow this procedure to address these problems:

1 Run the confidence test (see "Run Confidence Test on Configuration").

> **Tip** Run the confidence test as the first step in troubleshooting, as well as in validating your initial product installation and configuration.

2 If one or more tests fail, see the following information about the specific failure:

- "Test 1: Ping Using System Ping" on page 13-2
- "Test 2: Ping Using xpctargetping" on page 13-5
- "Test 3: Reboot Target Computer" on page 13-7
- "Test 4: Build and Download xpcosc" on page 13-9
- "Test 5: Check Host-Target Communications" on page 13-12
- "Test 6: Download Prebuilt Target Application" on page 13-14
- "Test 7: Execute Target Application" on page 13-15
- "Test 8: Upload Data and Compare" on page 13-16

3 Check the categorized questions and answers for clues to the root cause of the problem.

4 If the tests run, but task execution time is slow or the CPU overloads, see the questions and answers for Application Performance.

5 Check the MathWorks Support web site and MATLAB Central for tips. See "Where Is the MathWorks Support Web Site?" on page 25-2.

6 Call MathWorks Technical Support. See "How Do I Contact MathWorks Technical Support?" on page 25-5.

13

Confidence Test Failures

This topic describes guidelines, hints, and tips for questions or issues you might have while using the xPC Target product. Refer to the MathWorks Support xPC Target Web site (`http://www.mathworks.com/support/search_results.html?q=product:"xPC+Target"`) for specific troubleshooting solutions. The xPC Target documentation is also available from this site.

Test 1: Ping Using System Ping

If you are using a network connection, this test is a standard system ping to your target computer.

Note The confidence test skips test 1 for serial connections.

Troubleshoot failures with the following procedure:

1 Open a DOS shell and type the IP address of the target computer:

    ping xxx.xxx.xxx.xxx

Check the messages on your screen.

If DOS displays a message similar to the following, system ping succeeds even though test 1 fails.

    Pinging xxx.xxx.xxx.xxx with 32 bytes of data:
    Reply from xxx.xxx.xxx.xxx: bytes-32 time<10 ms TTL=59

If the DOS shell displays the following message, the system ping command failed.

    Pinging xxx.xxx.xxx.xxx with 32 byte of data:
    Request timed out.

2 **Ping succeeds — Ethernet addresses OK?**

If ping succeeds, check whether you entered the required IP and gateway addresses in xPC Target Explorer:

a Type xpcexplr in the MATLAB Command Window.

b In the **Targets** pane, expand the target computer node.

c Click the Target Properties icon in the toolbar or double-click **Properties**.

d Select **Host-to-Target communication**.

e Verify that **IP address**, **Subnet mask**, and **Gateway** boxes contain the required values.

f Select **Boot configuration**.

g Click **Create boot disk**.

h Reboot the target computer with the new kernel.

3 Ping fails — Cables OK?

If ping fails, first check your network cables. You might have a faulty network cable or, if you are using a coaxial cable, the terminators might be missing.

4 Ping fails — xPC Target properties OK?

Check that you have entered the required properties in xPC Target Explorer:

a Type xpcexplr in the MATLAB Command Window.

b In the **Targets** pane, expand the target computer node.

c Click the Target Properties icon in the toolbar or double-click **Properties**.

d Select **Host-to-Target communication**.

e Verify that **IP address**, **Subnet mask**, and **Gateway** boxes contain the required values.

f Verify that the bus settings match those of the target computer:

- For a PCI computer: check that **Bus type** is set to **PCI** instead of **ISA**.

- For an ISA computer:

  - Check that **Bus type** is set to **ISA** instead of **PCI**.

  - Check that **Address** is set to the required I/O port base address and that the address does not conflict with that of another hardware resource.

  - Check that **IRQ** is set to the required IRQ line and that the IRQ line does not conflict with that of another hardware resource.

- **–** If the target computer motherboard contains a PCI chip set, check whether the target computer BIOS reserves the IRQ line used by the ISA bus Ethernet card.

g Select **Boot configuration**.

h Click **Create boot disk**.

i Reboot the target computer with the new kernel.

5 Ping fails — Ethernet hardware operating?

Verify that your hardware is operating. For example, check that the green "ready" light goes on when the cable is connected to the Ethernet card.

6 Ping fails — Ethernet card supported?

Verify that you are using a supported Ethernet card on the target computer. See "Ethernet Communication Setup" for further details, including supplied Ethernet cards.

7 Ping fails — Not a locally mounted folder?

Run `xpctest` from a locally mounted folder, such as `Z:\work`, rather than from a UNC network folder, such as `\\Server\user\work`.

8 If these steps do not solve your problem, check the questions and answers for Host-Target Communication and section "Faulty BIOS Settings on Target Computer" on page 15-2.

9 If you still cannot solve your problem, see "How Do I Contact MathWorks Technical Support?" on page 25-5.

Test 2: Ping Using `xpctargetping`

This test is an xPC Target ping to your target computer. Troubleshoot failures with the following procedure:

1 In the MATLAB Command Window, type

```
tg=xpctarget.xpc('argument-list')
```

where `argument-list` is the connection information that indicates which target computer you are working with. If you do not specify arguments, the software assumes that you are communicating with the default target computer.

Check the messages in the MATLAB Command Window.

MATLAB should respond with the following messages:

```
xPC Object
    Connected           = Yes
    Application         = loader
```

2 Not connected — Bad target boot kernel?

If you do not get the preceding messages, you could have a bad target boot kernel. To solve this problem, create a new target boot kernel and reboot the target computer with the new kernel. See "Target Boot Methods".

3 Not connected — Environment variables set?

Use the PC MATLAB command to check the environment variables, in particular **Target PC IP address**. If test 1 passes but test 2 fails, you might not have entered the required IP address.

4 Not connected — Ethernet card supported?

If you are using a TCP/IP connection, make sure you are using a supported Ethernet card (see "Test 1: Ping Using System Ping" on page 13-2).

5 Not connected — RS-232 configuration?

If you are using an RS-232 connection, check the following:

- Verify that you are using a null modem cable (see "RS-232 Hardware").

- Verify that the COM ports on the host and target computers are enabled in the BIOS. If they are disabled, test 2 fails.

- Verify that the specified COM port is connected on each computer.

- Verify that the COM port being used matches the port specified in the target computer configuration.

Note RS-232 host-target communication mode will be removed in a future release. Use TCP/IP instead.

---

6 If these steps do not solve your problem, check the questions and answers for Host-Target Communication and section "Faulty BIOS Settings on Target Computer" on page 15-2.

7 If you still cannot solve your problem, see "How Do I Contact MathWorks Technical Support?" on page 25-5.

Test 3: Reboot Target Computer

This test tries to boot your target computer using an xPC Target command.

Note This procedure assumes that you have set environment settings with xPC Target Explorer. See "RS-232 Communication Setup" or "Ethernet Communication Setup".

Troubleshoot failures with the following procedure:

1 In the MATLAB Command Window, type

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

This command reruns the test without using the `xpctarget.xpc.reboot` command and displays the message

```
### Test 3, Software reboot the target PC: ... SKIPPED
```

2 **Build Succeeded — Software reboot supported?**

Check the results of `Test 4, Build and download an xPC Target application using model xpcosc`. If `xpctest` skips the `xpctarget.xpc.reboot` command but builds and loads the target application without producing an error message, the problem could be that the target computer does not support the xPC Target `reboot` command. In this case, you need to reboot using a hardware reset button.

3 **Build Failed — Kernel not loaded?**

If you saw the following error, the kernel might not be loaded when the host computer initiates communication with the target computer.

```
ReadFile Error: 6
```

Older xPC Target releases might receive this error. As a workaround, run `xpctest` with the `noreboot` option. For example,

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

This command runs the test without trying to reboot the target computer. It displays the following message:

```
### Test 3, Software reboot the target PC: ... SKIPPED
```

4 Build Failed — Example model modified?

If you directly or indirectly modify the `xpcosc` example model supplied with the product, test 3 is likely to fail.

Note Do not modify the files installed with the xPC Target software. If you want to modify one of these files, copy the file and modify the copy.

Restore the `xpcosc` example model to its original state by one of the following methods:

- Recreate the original model by editing it in the following location:

  matlabroot`\toolbox\rtw\targets\xpc\xpcdemos`

- Reinstall the software.

5 If these steps do not solve your problem, check the questions and answers for Target Computer Boot Process and section "Faulty BIOS Settings on Target Computer" on page 15-2.

6 If you still cannot solve your problem, see "How Do I Contact MathWorks Technical Support?" on page 25-5.

Test 4: Build and Download xpcosc

This test tries to build and download the model xpcosc. Troubleshoot failures with the following procedure:

1 In the MATLAB Command Window, check the error messages.

These messages help you locate where there is a problem.

2 Build Failed — Loader not ready?

If you get the following error message, reboot your target computer:

```
xPC Target loader not ready
```

This error message is sometimes displayed even if the target screen shows that the loader is ready.

3 Build Failed — Using full duplex?

If the communication between the host computer and target computer is TCP/IP, set the host computer network interface card (NIC) card and hub to half-duplex mode. Do not set the mode to full-duplex mode.

4 Build Failed — Compiler not supported?

Verify that a supported compiler is being used and that the blocks in the model can be compiled with the given compiler and compiler version.

5 Build Failed — Compiler path?

All Microsoft Visual compiler components must be in the Microsoft Visual Studio® folder after installation. If the compiler is not installed at the required location, you might get one of the following errors:

```
Error executing build command: Error using ==> make_rtw
Error using ==> rtw_c (SetupForVisual)
Invalid DEVSTUDIO path specified
```

or

```
Error executing build command: Error using ==> make_rtw
Error using ==> rtw_c
Errors encountered while building model "xpcosc"
```

along with the following MATLAB Command Window error:

```
NMAKE: fatal error U1064: MAKEFILE not found and no target
specified
Stop.
```

Verify your compiler setup:

a In the MATLAB command window, type:

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

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

```
Select your compiler for xPC Target.

[1] Microsoft Visual C++ Compilers 2008 Professional Edition (SP1) in
    c:\Program Files (x86)\Microsoft Visual Studio 9.0
[2] Microsoft Visual C++ Compilers 2010 Professional in
    C:\Program Files (x86)\Microsoft Visual Studio 10.0

[0] None

Compiler:
```

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

The function verifies your selection:

```
Verify your selection:

Compiler: Microsoft Visual C++ Compilers 2010 Professional
Location: C:\Program Files (x86)\Microsoft Visual Studio 10.0
```

```
Are these correct [y]/n?
```

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

The function finishes the dialog.

```
Done...
```

6 Build Failed — COM port read failed?

If you see the following MATLAB Command Window error:

```
ReadFile failed while reading from COM-port
```

- Check the state of your target computer. If it is unresponsive, you might need to reboot the target computer.

- In xPC Target Explorer, try to connect to the target computer again. Be sure to also check the connection between the host computer and target computer.

7 If these steps do not solve your problem, check the questions and answers for Model Compilation, Application Download, and Host-Target Communication and section "Faulty BIOS Settings on Target Computer" on page 15-2.

8 If you still cannot solve your problem, see "How Do I Contact MathWorks Technical Support?" on page 25-5.

Test 5: Check Host-Target Communications

This error occurs only when the environment variable settings are out of date. Troubleshoot failures with the following procedure:

1 Type xpcexplr in the MATLAB Command Window.

2 In the **Targets** pane, expand the target computer node.

3 Click the Target Properties icon in the toolbar or double-click **Properties**.

4 Select **Host-to-Target communication** and make the required changes to the communication properties.

Note RS-232 host-target communication mode will be removed in a future release. Use TCP/IP instead.

---

5 Select **Boot configuration**.

6 Set the required **Boot mode**.

Tip If you have xPC Target Embedded Option installed, verify that you have selected **Boot modeStand Alone**.

For information on boot options, see "Target Boot Methods".

---

7 Click **Create boot disk**

8 Reboot the target computer.

9 Rerun xpctest.

10 If these steps do not resolve the issue, recreate the target boot kernel using xpcbootdisk, reboot the target computer, and rerun xpctest.

11 If these steps do not solve your problem, check the questions and answers for Host-Target Communication and section "Faulty BIOS Settings on Target Computer" on page 15-2.

12 If you still cannot solve your problem, see "How Do I Contact MathWorks Technical Support?" on page 25-5.

Test 6: Download Prebuilt Target Application

This test runs the basic target object constructor, xpc. This error rarely occurs unless an earlier test has failed.

1 Verify that the preceding steps completed without producing an error message.

2 Configure, build and download the tutorial model and record whatever error messages appear (see "Build and Download Target Application").

3 If these steps do not solve your problem, check the questions and answers for Application Download and Host-Target Communication and section "Faulty BIOS Settings on Target Computer" on page 15-2.

4 If you still cannot solve your problem, see "How Do I Contact MathWorks Technical Support?" on page 25-5.

Test 7: Execute Target Application

This test executes a target application (xpcosc) on the target computer. This test fails if you change the xpcosc model start time to something other than 0, such as 0.001. This change causes the test, and the MATLAB interface, to halt. To address this failure:

1 Set the xpcosc model start time back to 0.

2 Rerun the test.

3 If these steps do not solve your problem, check the questions and answers for Application Execution, Application Performance, Application Signals, and Application Parameters and section "Faulty BIOS Settings on Target Computer" on page 15-2.

4 If you still cannot solve your problem, see "How Do I Contact MathWorks Technical Support?" on page 25-5.

Test 8: Upload Data and Compare

This test executes a target application (xpcosc) on the target computer. This test might fail if you change the xpcosc model (for example, if you remove the Outport block).

Note Do not modify the files installed with the xPC Target software. If you want to modify one of these files, copy the file and modify the copy.

---

1 To eliminate this problem, restore the xpcosc example model to its original state by one of the following methods:

- Recreate the original model by editing it in the following location:

  matlabroot\toolbox\rtw\targets\xpc\xpcdemos

- Reinstall the software.

2 Other issues might also cause this test to fail. If you still need more help, check the following:

- If you are running a new xPC Target release, be sure that you have a new target boot kernel for this release. See "What Should I Do After Updating Software?" on page 25-4.

- There is a known issue with xPC Target software version 1.3. It might occur when you run xpctest two consecutive times. See the known issue and solution documented in http://www.mathworks.com/support/solutions/data/1-18DTB.html.

3 If you are installing another version of the xPC Target software on top of an existing version, check the version number of the current installation. At the MATLAB command line, type xpclib. The version number appears at the bottom of the xPC Target block library window. If the version number is not the one to which you want to upgrade, reinstall the software.

4 If these steps do not solve your problem, check the questions and answers for Application Execution, Application Performance, Application Signals, and Application Parameters and section "Faulty BIOS Settings on Target Computer" on page 15-2.

5 If you still cannot solve your problem, see "How Do I Contact MathWorks Technical Support?" on page 25-5.

Host Computer Configuration

Why Does Boot Drive Creation Halt?

If your host computer MATLAB interface halts while creating an xPC Target boot disk or network boot image:

- Use another drive to create a new xPC Target boot drive or network boot image.

- If your host computer has antivirus software, it might conflict with the MATLAB software. Disable the software while using the MATLAB interface.

- Verify that the host computer drive is accessible. If it is not accessible, replace the drive.

Target Computer Configuration

Faulty BIOS Settings on Target Computer

The BIOS settings of a computer system influence how the computer works. If you experience problems using the xPC Target software, check the system BIOS settings of the target computer. These settings are beyond the control of the xPC Target product. See "Target Computer BIOS Settings".

Faulty BIOS settings can cause issues like the following:

- Why is my target not booting?
- Why can `getxpcpci` detect PCI boards, but `autosearch -l` cannot?
- Why can my standalone application run on some target computers, but not others?
- Why is my target computer crashing while downloading applications?
- Why is my target PC104 hanging on boot?
- Why is my boot time slow?
- Why is my software not running in real time?
- Why are my USB ports not working?

Allowable Partitions on the Target Hard Drive

The target computer hard drive can contain one or multiple partitions. However, the xPC Target software supports file systems of type FAT-12, FAT-16, or FAT-32 only.

File System Disabled on the Target Computer

If your target computer does not have a FAT hard disk, the monitor on the target computer displays the following error:

```
ERROR -4: drive not found
No accessible disk found: file system disabled
```

If you do not want to access the target computer file system, you can ignore this message. If you want to access the target computer file system, add a FAT hard disk to the target computer system and reboot.

Tip Verify that the hard drive is not cable-selected and that the BIOS can detect it.

Adjust the Target Computer Stack Size

To discover and adjust the stack size used by the real-time threads on the target computer:

1 Add the following blocks to your model:

- xPC Target Get Free Stack Size — Outputs the number of bytes of stack memory currently available to the target application thread.

- xPC Target Get Minimal Free Stack Size — Outputs the number of bytes that have not been used in the stack since the thread was created.

Note The underlying function traverses the entire stack to find unused bytes. This is a time-consuming operation. Therefore, Get Minimal Free Stack Size should be used only for diagnostic purposes.

---

2 Execute the target application, monitoring the stack size and minimal stack size.

3 Calculate a stack size that allows execution to proceed.

Note

- To meet the memory requirements, you might have to reconfigure your target computer.

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

---

4 Adjust the stack size of the real-time threads by setting a TLC option in the Configuration Parameters dialog, Code Generation node, section Build Process.

For example, to set the stack size to 256 kBytes, type the following in the TLC option box:

```
-axPCModelStackSizeKB=256
```

How Can I Get PCI Board Information?

Information about the PCI devices in your target computer is useful if you want to determine what PCI boards are installed in your xPC Target system, or if you have multiple boards of a particular type in your system. Before you start, determine what boards are installed in your target computer by typing the following in the MATLAB Command Window:

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

Note Typing this command will automatically connect the host computer to the default target computer, if it is running.

If you have or want to use multiple boards of a particular type in your system, verify that the I/O driver supports multiple boards. See the "Multiple board support" entry for this board type in the xPC Target library or the xPC Target Interactive Hardware Selection Guide (`http://www.mathworks.com/support/product/XP/productnews /interactive_guide/xPC_Target_Interactive_Guide.html`.

If you confirm that the board type supports multiple boards, and these boards are installed in the xPC Target system, do the following to obtain the bus and slot information for these boards:

1 In the PCI devices display, note the contents of the `Bus` and `Slot` columns of the PCI devices in which you are interested.

2 Enter the bus and slot numbers as vectors into the **PCI Slot** parameter of the PCI device. For example:

```
[1 9]
```

where `1` is the bus number and `9` is the slot number.

For additional information about PCI bus I/O devices, refer to "PCI Bus I/O Devices".

How Do I Diagnose My Board Driver?

If you encounter issues using the xPC Target I/O drivers:

1 Display the input/output behavior of the board using an external instrument, such as an oscilloscope or logic analyzer.

2 Verify that you have configured the I/O board driver according to the manufacturer's data sheet.

3 Verify that you are using the latest version of the I/O board driver and of the xPC Target software. See "How Do I Get a Software Update?" on page 25-3.

4 Verify that the behavior persists when you run the target application on a different target computer.

5 Verify that the behavior persists when you install another instance of the I/O board in the target computer.

6 Download the manufacturer's I/O driver and diagnostic software from the manufacturer web site, install the driver and software on your computer, and test the hardware using the manufacturer's software.

7 Report the issue to MathWorks Support at `http://www.mathworks.com/support/contact_us/index.html`.

16

Host-Target Communication

- "Is There Communication Between the Computers?" on page 16-2
- "Boards with Slow Initialization" on page 16-4
- "Timeout with Multiple Ethernet Cards" on page 16-6
- "Recovery from Board Driver Errors" on page 16-8
- "How Can I Diagnose Network Problems?" on page 16-9

Is There Communication Between the Computers?

Use the following MATLAB commands from the host computer to validate the host/target setup:

- xpctargetping

  The xpctargetping command performs a basic communication check between the host and target computers. This command returns success only if the xPC Target kernel is loaded and running and the host and target computer are communicating. Use this command for a quick check of the communication between the host computer and target computer.

- xpctest

  The xpctest command performs a series of tests on your xPC Target system. These tests range from performing a basic communication check to building and running target applications. At the end of each test, the command returns an OK or failure message. If the test is inappropriate for your setup, the command returns a SKIPPED message. Use this command for a thorough check of your xPC Target installation.

Communication errors might also occur in the following instances:

- The target computer is running an old xPC Target boot kernel that is not in sync with the xPC Target release installed on the host computer. Create a new target boot kernel for each new release.

- If the communication between the host computer and target computer is TCP/IP, set the host computer network interface card (NIC) card and hub to half-duplex mode. Do not set the mode to full-duplex mode.

- If you have an active firewall in your system, you might experience communication errors. For example, build errors might occur if you try to build and download a model with a thermocouple board (causing a slower initialization time) in a system that contains a firewall. To work around this issue, you can add the MATLAB interface to the firewall exception list. See also "Boards with Slow Initialization" on page 16-4

- To diagnose BIOS problems, see:
  - "Faulty BIOS Settings on Target Computer" on page 15-2
  - "Target Computer BIOS Settings"

- If multiple Ethernet cards or chips are installed in the target computer, see
  "Timeout with Multiple Ethernet Cards" on page 16-6.

Boards with Slow Initialization

Some xPC Target boards take a long time to initialize. This situation might cause the software to run out of time before a model downloads, causing the host computer to disconnect from the target computer.

By default, if the host computer does not get a response from the target computer after downloading a target application and waiting 5 seconds, the host computer software times out. The target computer responds only after downloading and initializing the target application.

Usually 5 seconds is enough time to initialize a target application, but in some cases it might not be long enough. The time to download a target application mostly depends on your I/O hardware. For example, thermocouple hardware (such as the PCI-DAS-TC board) takes longer to initialize. With slower hardware, you might also get errors when building and downloading an associated model. Even though the target computer is fine, a false timeout is reported and you might get an error like the following:

`"cannot connect to ping socket"`

This is not a fatal error. You can reestablish communication with the following procedure:

1 Type `xpctargetping` at the MATLAB command prompt.

2 Wait for the system to return from the `xpctargetping` command. If `xpctargetping` finds a working connection between the host computer and target computer, the response is something like:

```
ans =

success
```

3 Restart the target object.

Alternatively, you can increase the timeout value, using the following procedure:

1 In your Simulink model, select **Simulation > Model Configuration Parameters**, and navigate to the **xPC Target options** node.

2 Clear the **Use default communication timeout** parameter.

The **Specify the communication timeout in seconds** parameter appears.

3 Specify a new timeout value, in seconds. For example, enter 20 in parameter **Specify the communication timeout in seconds**.

4 Click **OK**.

5 In the Simulink Editor window and from the **Code** menu, click **C/C++ Code > Build Model**.

In this case, the host computer waits for about 20 seconds before declaring that a timeout has occurred. It does not take 20 seconds for every download. The host computer polls the target computer about once every second, and if a response is returned, returns the success value. Only in the case where a download really fails does it take the full 20 seconds.

Timeout with Multiple Ethernet Cards

The xPC Target product supports a number of Ethernet cards and chips, as described in "Ethernet Communication Setup". If your target computer has more than one of these cards or chips installed, you could experience timeout problems. For example, suppose you are using the Network Boot option to boot the target computer. If the host computer boots the target computer using Ethernet A on the target computer, it associates the IP address of the target computer with the Media Access Control (MAC) address of Ethernet adapter A. If, after it does so, the target computer BIOS connects the target computer to Ethernet B, the xPC Target software cannot connect the host and target computers because they are connected to different Ethernet controllers.

First, try to disable or remove the Ethernet controller that you will not use. For example, if you have both an on-board Ethernet controller and a separate Ethernet card, you could disable the on-board Ethernet controller through the target computer BIOS. If you are required to have multiple Ethernet adapters of the same type in the target computer, you might need to experiment to determine which Ethernet adapter the software has chosen.

If you are not using the Network Boot option to boot the target computer and cannot establish communication between the target computer and host computer:

1 Switch the network cable to the other Ethernet port and try again.

2 If you can establish communication, use this Ethernet port to connect the host computer to the target computer.

If you are using the Network Boot option and experience this issue, do the following:

1 Connect the network cable to Ethernet adapter B.

2 In the MATLAB Command Window, type

```
!arp -d
```

This command removes the association between the target computer address and the hardware address of Ethernet adapter A from the cache of the host computer. This removal allows a new connection (and association) to be made.

3 Change the Ethernet adapter card that the Network Boot option uses. You can do this in one of the following ways:

- Change the target computer BIOS to change the Ethernet adapter to the one that the Network Boot option is looking for.

- Follow the procedure "Ethernet Card Selection by Index" on page 4-27..

Recovery from Board Driver Errors

If an error in a driver causes the xPC Target system to crash, a timeout occurs and `xpctargetping` fails with an error message. In such an event, you need to reboot the target and reestablish communication between the host computer and target computer.

To get the xPC Target system back up and running:

1 Remove the reference to the problem driver from the model.

2 Reboot the target computer.

3 At the MATLAB command line, issue `xpctargetping` to reestablish communications.

4 If the driver with which you are having problems is one provided by MathWorks, try to pinpoint the problem area (for example, determine whether certain settings in the driver block cause problems).

Alternatively, you can exit and restart the MATLAB interface.

How Can I Diagnose Network Problems?

If you experience network problems when using this product,
use an available computer with Internet access to
refer to the MathWorks Support xPC Target Web site
(`http://www.mathworks.com/support/search_results.html?q=product:"xPC+Target`
This Web site has the most up-to-date information about this topic.

Target Computer Boot Process

- "Why Won't the Target Computer Boot?" on page 17-2
- "Why Won't the Kernel Load?" on page 17-4
- "Why Is the Target Medium Not Bootable?" on page 17-5
- "Why Is the Target Computer Halted?" on page 17-6

Why Won't the Target Computer Boot?

If your target computer cannot boot with the xPC Target boot disk, removable boot drive, or network boot image:

- Recreate the target boot kernel using new media.

- Verify using xpcgetenv that the current properties in the xPC Target kernel correspond with the environment variables displayed in the **Host-to-Target communication** and **Target settings** panes of xPC Target Explorer.

Tip To display the allowed values of xPC Target environment properties, type setxpcenv without arguments. To display their current values, type getxpcenv without arguments.

- Verify that the xPC Target boot disk or removable boot drive contains files like the following:
  - BOOTSECT.RTT
  - XPCTGB1.RTA

Note The name of the last file varies depending on the communication method.

- If the .RTT and .RTA files are not complete, reinstall the software.

- The xPC Target kernel may not be able to discover system hardware not compliant with the Advanced Configuration and Power Interface (ACPI) standard. To allow the kernel to discover such hardware, use the following xPC Target environment property to access the legacy MPFPS in the computer BIOS:

  ```
  setxpcenv('LegacyMultiCoreConfig', 'on')
  ```

- If you are doing a network boot and the boot procedure displays a message similar to TFTP Timeout:

- Verify that the `xpctftpserver` program is running. If it is not, recreate the network boot image.

- Temporarily disable the Internet security (firewall) software on the host computer. If you can now boot:

  · Follow the Internet security software instructions to allow the xPC Target boot procedure to work in its presence. For example, add the MATLAB interface to the firewall exception list.

  · Reenable the Internet security software.

• If problems persist, see the questions and answers for Target Computer Boot Process.

• If you still cannot boot the target computer from a boot disk or removable boot drive, you might need to replace the target computer disk drive hardware.

Why Won't the Kernel Load?

When booting the target computer, you might see a message like the following:

```
xPC Target 4.X loading kernel..@@@@@@@@@@@@@@@@@@@@@@@@
```

The target computer displays this message when it cannot read and load the kernel from the target boot disk.

The probable cause is a bad boot kernel. To diagnose this problem, recreate the target boot kernel. If you have a removable boot drive, reformat the drive or use a new formatted drive. If you have a boot CD, create a new boot disk. If you are using network boot, recreate the network boot image.

Why Is the Target Medium Not Bootable?

When booting the target computer, you might get a message similar to the following:

```
Not a bootable medium or NTLDR is missing
```

Selecting either **DOS Loader** or **Stand Alone** mode instead of **Removable Disk** mode can cause this message.

To solve this problem:

1 Type xpcexplr in the MATLAB Command Window.

2 In the **Targets** pane, expand the target computer node.

3 Click the Target Properties icon in the toolbar or double-click **Properties**.

4 Select **Boot configuration** and select the desired entry in the **Boot mode** list.

5 Click **Create boot disk.**.

Why Is the Target Computer Halted?

If your target computer displays a System Halted message while booting:

- Verify in the **Host-to-Target communication** pane of xPC Target Explorer that the **Target driver** parameter is configured as required by your network.

- Recreate the target boot kernel using new media and use the new kernel to boot the target computer.

- Verify that the xPC Target software supports your target computer hardware. Be sure to verify the network communication hardware.

Modeling

- "How Do I Handle Encoder Register Rollover?" on page 18-2
- "How Can I Write Custom Device Drivers?" on page 18-3

How Do I Handle Encoder Register Rollover?

Encoder boards have a fixed size counter register of 16 bits, 24 bits, or 32 bits. Regardless of the size, the register eventually overflows and rolls over. Registers can roll over in either the positive or negative direction.

Some boards provide a hardware mechanism to account for overflows or rollovers. As a best practice, you should design your model to deal with overflows or rollovers. Defining an initial count can handle the issue for some applications.

To handle register rollovers, you can use standard Simulink blocks to design the following counter algorithm types:

- Rollover Counter — Counts the number of times the output of an encoder block has rolled over. This counter should count up for positive direction rollovers and down for negative direction rollovers.

- Extended Counter — Provides a rollover count not limited by register size. For an n-bit register, this counter should be able to count values greater than $2^{(n-1)}$.

The Incremental Encoder/Utilities/Rollover sublibrary of the xPC Target library contains example blocks for these two types of counters. See Rollover Counter and Extended Counter for further details. You can use these blocks in your model as is, or modify them for your model. Connect the output of the encoder block to these blocks.

Note To view the algorithms used in these implementations, right-click the subsystem and select **Mask > Look Under Mask**.

Keep the following requirements in mind when using these blocks:

- Some driver blocks allow an initial starting value to be loaded into the register. You must pass this value to the rollover blocks to adjust for that offset.

- The rollover block needs to know how many counts each rollover represents. Typically, this number is 2^n, where n is the size of the register in bits.

How Can I Write Custom Device Drivers?

You might want to write your own driver if you want to include an unsupported device driver in your xPC Target system. See "Custom Device Drivers".

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 the data required by the model or S-function, including block parameters and outputs.

- An excellent understanding of the I/O hardware. Because of the real-time nature of the xPC Target system, you must develop drivers with minimal latency. Because most drivers access the I/O hardware at the lowest possible level (register programming), you must have a good understanding of how to control a board with register information and have access to the register-level programming manual for the device.

- A good knowledge of port and memory I/O access over various buses. You need this information to access I/O hardware at the register level.

19

Model Compilation

Requirements for Standalone Target Applications

You can use either the xPC Target API dynamic link library (DLL) or the xPC Target component object model (COM) API library to create a custom standalone interface to control a real-time application running on the target computer. To deploy these standalone applications, you must have the xPC Target Embedded Option license. Without this license, you can create and use the standalone application in your environment, but you cannot deploy that application on another host computer that does not contain your licensed copy of the xPC Target software.

See "Stand Alone Boot Method".

Compiler Errors from Models Linked to DLLs

The xPC Target software supports links to static link libraries (.lib) **only**, not links to dynamic link libraries (.dll). When you compile your models, verify that you link only to static link libraries. Linking to static libraries is not an issue when you compile with xPC Target S-functions.

Compilation Failure with WATCOM Compilers

The Open WATCOM compiler is no longer supported. Use a Microsoft compiler instead.

Application Download

- "Why Does My Download Time Out?" on page 20-2
- "Increase the Time for Downloads" on page 20-4
- "Why Does the Download Halt?" on page 20-5

Why Does My Download Time Out?

If the host computer and target computer are not connected, or you have not entered the required environment properties, the download process terminates after about 5 seconds with a timeout error. Be sure that you have followed the instructions outlined in "Host-Target Configuration" before continuing.

To diagnose the problem, use the following procedure:

1 Type xpcexplr in the MATLAB Command Window.

2 In the **Targets** pane, expand the target computer node.

3 Click the Target Properties icon 🖼 in the toolbar or double-click **Properties**.

4 Select **Host-to-Target communication** and make the required changes to the communication properties.

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

5 Select **Boot configuration** and click **Create boot disk**.

6 Reboot the target computer and try downloading the application again.

7 In some cases, the download might have completed even though you get a timeout error. To detect this condition, wait until the target screen displays

```
System:initializing application finished.
```

8 Type xpctargetping at the MATLAB command prompt.

If xpctargetping finds a working connection between the host computer and target computer, the response is something like:

```
ans =
```

```
success
```

9 Right-click the target computer in question and select **Connect**.

If the connection resumes, the connection is working. If the connection times out consistently for a particular model, the timeout needs to be increased. See "Increase the Time for Downloads" on page 20-4.

For information on setting up the xPC Target environment, see either "RS-232 Settings" or "ISA Bus Ethernet Settings", and then see "Target Boot Methods".

Increase the Time for Downloads

By default, if the host computer does not get a response from the target computer after downloading a target application and waiting about 5 seconds, the host computer software times out. On the other hand, the target computer responds only after downloading and initializing the target application.

Usually 5 seconds is enough time to download a target application, but in some cases it may not be long enough. The time to download a target application mostly depends on your I/O hardware. For example, thermocouple hardware takes longer to initialize. In this case, even though the target computer is fine, a false timeout is reported.

You can increase the timeout value in one of the following ways:

- At the model level, open the **Simulink > Model Configuration Parameters** dialog box and navigate to the **xPC Target options** node. Clear the **Use default communication timeout** parameter and enter a new desired timeout value in the **Specify the communication timeout in seconds** parameter. For example, enter 20 to increase the value to 20 s.

- At the target application level, use the target application xpctarget.xpc.set (target application object) method to set the CommunicationTimeOut property to the desired timeout value. For example, to increase the value to 20 s:

  ```
  tg.set('CommunicationTimeOut',20)
  ```

For both methods, the host computer polls the target computer about once every second, and if a response is returned, returns the success value. Only if a download really fails does the host computer wait the full twenty seconds.

Why Does the Download Halt?

If the MATLAB interface freezes and there are target ping errors, this failure is likely the result of an active firewall, a long initialization process, or both combined. To diagnose this problem, see:

- "Is There Communication Between the Computers?" on page 16-2
- "Boards with Slow Initialization" on page 16-4

  "Timeout with Multiple Ethernet Cards" on page 16-6

21

Application Execution

View Application Execution from the Host

xPC Target displays output from the target application on the target computer monitor. You can view this monitor from the host computer using Real-Time xPC Target Spy.

For a single-target system, type:

```
xpctargetspy
```

For a particular target computer `TargetPC1`, type:

```
xpctargetspy('TargetPC1')
```

The xPC Target Spy window is displayed on the host computer monitor.

Sample Time Deviates from Expected Value

You might notice that the sample time you measure from your model is not equal to the sample time you requested. This difference depends on your hardware. Your model sample time is as close to your requested time as the hardware allows.

However, hardware does not allow infinite precision in setting the spacing between the timer interrupts. This limitation can cause the divergent sample times.

For the supported target computers, the only timer that can generate interrupts is based on a 1.193 MHz clock. For the xPC Target system, the timer is set to a fixed number of ticks of this frequency between interrupts. If you request a sample time of 1/10000 seconds, or 100 microseconds, you do not get exactly 100 ticks. Instead, the xPC Target software calculates that number as:

$$100 \times 10^{-6} \text{ s} \times 1.193 \times 10^{6} \text{ ticks/s} = 119.3 \text{ ticks}$$

The xPC Target software rounds this number to the nearest whole number, 119 ticks. The actual sample time is then:

$$119 \text{ ticks}/(1.193 \times 10^{6} \text{ ticks/s}) = 99.75 \times 10^{-6} \text{ s}$$
$$(99.75 \text{ microseconds})$$

Compared to the requested original sample time of 100 microseconds, this value is 0.25% faster.

As an example of how you can use this value to derive the expected deviation for your hardware, assume the following:

- Output board that generates a 50 Hz sine wave (expected signal)
- Sample time of 1/10000
- Measured signal of 50.145 Hz

The difference between the expected and measured signals is 0.145 Hz, which deviates from the expected signal value by 0.29% (`0.145 / 50`). Compared

to the previously calculated value of 0.25%, there is a difference of 0.04% from the expected value.

If you want to further refine the measured deviation for your hardware, assume the following:

- Output board that generates a 50 Hz sine wave (expected signal)
- Sample time of 1/10200
- Measured signal of 50.002 Hz

  ```
  1/10200 s X 1.193 x 10⁶ ticks/s = 116.96 ticks
  ```

Round this number to the nearest whole number of 117 ticks. The resulting frequency is then

```
(116.96 ticks/117)(50) = 49.983 Hz
```

The difference between the expected and measured signal is 0.019, which deviates from the expected signal value by 0.038% (`0.019 / 50.002`). The deviation when the sample time is 1/10000 is 0.04%.

Some amount of error is common for most computers, and the margin of error varies from machine to machine.

Note Most high-level operating systems, like Microsoft Windows or Linux®, occasionally insert extra long intervals to compensate for errors in the timer. Be aware that the xPC Target software does not attempt to compensate for timer errors. For this product, close repeatability is more important for most models than exact timing. However, some chips might have inherent designs that produce residual jitters that could change your system behavior. For example, some Intel® Pentium chips might produce residual jitters on the order of 0.5 microsecond from interrupt to interrupt.

What Measured Sample Time Can I Expect?

The xPC Target kernel is tuned for minimal overhead and maximum performance. To check what sample time you can expect, run xpcbench at the MATLAB command line.

- xpcbench('this') — Evaluates your target computer against predefined benchmarks and compares it to other target computers. The results indicate the smallest base sample time that an xPC Target application can achieve on your system.
- xpcbench('model') — Evaluates your target computer against your specific model.

Actual obtainable sample times depend on a number of factors, including:

- Processor performance
- Model complexity
- I/O block types
- Number of I/O channels

Why Has the Stop Time Changed?

If you change the step size of a target application after it has been built, it is possible that the target application will execute for fewer steps than you expect. The number of execution steps is:

```
floor(stop time/step size)
```

When you compile code for a model, Simulink Coder calculates a number of steps based on the current step size and stop time. If the stop time is not an integral multiple of the step size, Simulink Coder adjusts the stop time for that model based on the original stop time and step size. If you later change a step size for a target application but do not recompile the code, xPC Target uses the new step size and the previously adjusted stop time. The resulting model may execute for fewer steps than you expect.

For example, if a model has a stop time of 2.4 and a step size of 1, Simulink Coder adjusts the stop time of the model to 2 at compilation. If you change the step size to 0.6 but do not recompile the code, the expected number of steps is 4, but the actual number of steps is 3 because xPC Target uses the previously adjusted stop time of 2.

To avoid this problem, verify that the original stop time (as specified in the model) is an integral multiple of the original step size.

Why Is the Web Interface Not Working?

The Web interface to the target computer requires a connection between a Web browser and the IP address and port by which you access the target. If this IP address and port is already in use because you connected to the target via Simulink, xPC Target Explorer, or a MATLAB command such as `xpc`, the Web interface cannot connect and will fail.

Tip Type the MATLAB command `xpcwwwenable` immediately before opening the Web interface.

22

Application Parameters

Why Does the getparamid Function Return Nothing?

The `xpctarget.xpc.getparamid` and `xpctarget.xpc.getsignalid` functions accept `block_name` parameters. For these functions, enter for `block_name` the mangled name that the Simulink Coder software uses for code generation. You can determine the `block_name` as follows:

- If you do not have special characters in your model, use the `gcb` function.

- If the blocks of interest have special characters, retrieve the mangled name with `tg.showsignals='on'` or `tg.showparam = 'on'`.

For example, if carriage return `'\n'` is part of the block path, the mangled name returns with carriage returns replaced by spaces.

Which Model Parameters Can I Tune?

You can tune parameters of fixed-point data types, such as Boolean, integer, and double. For more on fixed-point data types, see "Supported Data Types"

You cannot tune parameters of complex or multiword data types.

Application Signals

How Do I Fix Invalid File IDs?

You might get `Error -10:  Invalid File ID` on the target computer if you are acquiring signal data with a file scope. This error occurs because the size of the signal data file exceeds the available space on the disk. The signal data is most likely corrupt and irretrievable. You should delete the signal data file and reboot the xPC Target system. To prevent this occurrence, monitor the size of the signal data file as the scope acquires data.

Refer to the MathWorks Support xPC Target Web site (`http://www.mathworks.com/support/search_results.html?q=product:"xPC+Target`) for additional information.

Which Model Signals Can I Access?

You cannot directly access or tag signals from virtual buses or blocks. To observe a virtual block:

1 Add a unity `Gain` block (a Gain block with a gain of `1.0`) to the model.

2 Connect the signal output of the virtual block to the input of the unity `Gain` block.

3 Access or tag the output signal of the unity `Gain` block.

To observe a virtual bus, add a unity `Gain` block to each individual signal.

You cannot directly access signals you have optimized with block reduction optimization. Access these signals by making them test points.

You cannot access signals of complex or multiword data types.

24

Application Performance

How Can I Improve Run-Time Performance?

To improve runtime performance and reduce the task execution time (TET) of a model `model`:

1 Run `xpcbench` at the MATLAB command line:

- `xpcbench('this')` — Evaluates your target computer against predefined benchmarks and compares it to other target computers. The results indicate the smallest base sample time that an xPC Target application can achieve on your system.

- `xpcbench('model')` — Evaluates your target computer against your specific model.

Tip For more information on xPC Target benchmarking, see `http://www.mathworks.com/support/product/XP/-productnews/benchmarks.html`.

---

2 If your target computer is not high on the list of benchmark computers, consider switching to a target computer with higher performance.

3 Run the xPC Target profiler on `model` and record where the time is being spent. (See "Profiling Target Application Execution" on page 26-6.)

4 If the model contains many states (for example, more than 20 states), clear the **States** check box in the **Data Import/Export** pane of the Configuration Parameters dialog box. This disables state logging, making more memory available for the target application. An alternative to logging states is to select individual states of interest by adding Outport blocks to the model.

5 Clear the **Save to workspace** check boxes in the **Data Import/Export** pane of the Configuration Parameters dialog box (**Time**, **States**, **Output**, **Final states**, **Signal logging**). This turns logging off, making more computing time available for calculating the model.

6 Clear the **Log Task Execution Time** check box in the **xPC Target options** pane of the Configuration Parameters dialog box. This disables TET logging for the application.

7 Increase **Fixed-step size (fundamental sample time)** in the **Solver** pane of the Configuration Parameters dialog box. Executing with a very short sample time might overload the CPU.

8 Use polling mode, if you do not need background processes (see "Polling Mode" on page 6-5 for more on setting this mode).

9 Disable the target scope display. To do this, clear the **Graphics mode** check box in the **Target settings** pane of xPC Target Explorer.

10 Use fewer scopes in the model.

11 Reduce the number of I/O channels in the model.

12 Consider partitioning the model and running it on a multicore system (see "Design Considerations").

> **Note** To use your target computer in multicore mode, you must set the **Multicore CPU** check box in the **Target settings** pane of xPC Target Explorer.

13 Consider partitioning the model and running it on multiple target computers. This optimization might require multitarget synchronization using CAN, UDP, parallel port, or reflective memory.

14 Check the questions and answers under Application Performance for tips on eliminating CPU overloads and improving task execution time.

15 Check the MathWorks Support Web site and MATLAB Central for other tips. See "Where Is the MathWorks Support Web Site?" on page 25-2.

16 Call MathWorks Technical Support. See "How Do I Contact MathWorks Technical Support?" on page 25-5.

Why Does Model Execution Produce CPU Overloads?

A CPU overload indicates that the CPU was unable to complete processing a model time step before being asked to restart. When an overload occurs, one of the following can happen:

- The xPC Target kernel halts model execution.

- If the overload is allowed, the model execution continues until a predefined event (see "Can I Allow CPU Overloads?" on page 24-7 for details). If a model continues running after a CPU overload, the model time step is as long as the time required to finish the execution. This behavior delays the following time step.

This error might occur if you have:

- **Real CPU overloads** — Those caused by model design or target computer resources. For example, a model is trying to do more than can be done in the allocated time on the target computer. Possible reasons are:

  - The target computer is too slow or the model sample time is too small (see "How Small Can the Sample Time Be?" on page 24-6).

  - The model is too complex (algorithmic complexity).

  - I/O latency, where each I/O channel used introduces latency into the system. This might cause the execution time to exceed the model time step.

    To address I/O latency, you can use the xPC Target Interactive Guide (`http://www.mathworks.com/support/product/XP/productnews/-interactive_guide/xPC_Target_Interactive_Guide.html`) to find latency numbers for boards supported by the block library. For example, if your application includes the National Instruments® PCI-6713 board, and you want to use four outputs:

    1 Look up the board in the xPC Target Interactive Guide.

    From the table, the D/A latency is `1+2.4N`.

    2 To get the latency for four outputs, calculate the latency

    1+(2.4 x 4) = 10.6 microseconds

3 Include this value in your sample time calculations.

- **Spurious CPU overloads** — Commonly caused by factors outside of the model design. These overloads are most likely caused by one of the following:

  - Advanced Power Management

  - Plug-and-Play (PnP) operating system

  - System Management Interrupts (SMIs)

  Enabling these properties causes non-real-time behavior from the target computer. You must disable these BIOS properties for the target computer to run the target application in real time. See"Target Computer BIOS Settings".

Note Some BIOS do not allow you to disable SMIs.. However, for some chipsets, you can programmatically prevent or disable SMIs. For example, see the *Disabling SMIs on Intel ICH5 Chipsets* document at MATLAB Central for a solution to disabling SMIs in the Intel ICH5 family.

For further information and test models, see the *xPC Target CPU Overloads* document at MATLAB Central.

How Small Can the Sample Time Be?

If the model has too small a sample time, a CPU overload can occur. This error indicates that to run the target application, executing one step of the model requires more time than the sample time for the model (Fixed step size property) allows.

When this error occurs, the target object property CPUoverload changes from none to detected. To diagnose the issue:

1 Run xpcbench at the MATLAB command line:

- xpcbench('this') — Evaluates your target computer against predefined benchmarks and compares it to other target computers. The results indicate the smallest base sample time that an xPC Target application can achieve on your system.

- xpcbench('model') — Evaluates your target computer against your specific model.

Tip For more information on xPC Target benchmarking, see http://www.mathworks.com/support/product/XP/-productnews/benchmarks.html.

2 Change the model Fixed step size property to at least the indicated value and rebuild the model. Use the **Solver** node in the Simulink model Configuration Parameters dialog.

3 If these steps do not solve your problem, see:

"How Can I Improve Run-Time Performance?" on page 24-2.

Can I Allow CPU Overloads?

Typically, the xPC Target kernel halts model execution when it encounters a CPU overload. You can direct the xPC Target environment to allow CPU overloads using the following options in the **TLC options** parameter in the Code Generation pane of the Simulink Configuration Parameters dialog box.

Option	Description	Default
xPCMaxOverloads	Number of acceptable overloads.	0
xPCMaxOverloadLen	Number of contiguous acceptable overloads. If you do not specify this option, the default value is the same as xPCMaxOverloads. Specify a value that is the same or less than the value for the xPCMaxOverloads option. Do not a use a value greater than xPCMaxOverloads.	Same as value of xPCMaxOverloads
xPCStartupFlag	Number of executions of the model at startup, where the timer interrupt is temporarily disabled during model execution. After the model finishes the first xPCStartupFlag number of executions, the xPC Target software enables the timer interrupt, which will invoke the next execution for the model.	1

If you experience a CPU overload after the model starts, the software ignores timer interrupts if the task is already running. The model continues running, subject to the values of xPCMaxOverloads and xPCMaxOverloadLen. The model then executes at the next step.

Consider the following cases:

- `xPCMaxOverloads` is 2. The software tolerates the first two overloads and stops execution at the third.
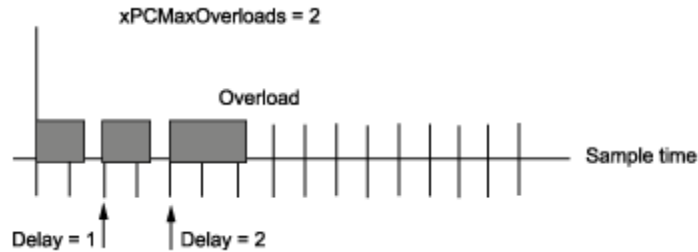
- `xPCMaxOverloads` is 3 and `xPCMaxOverloadLen` is 2. The software tolerates the first three overloads and halts the model at the fourth.
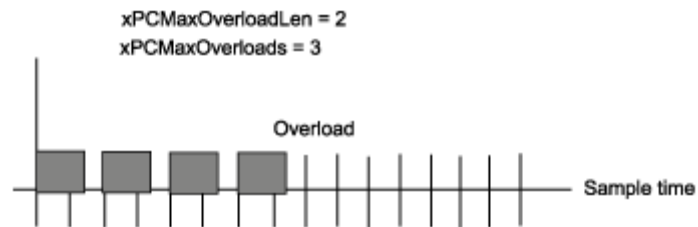
- `xPCStartupFlag` is 4. The kernel ignores overloads for the first four executions.
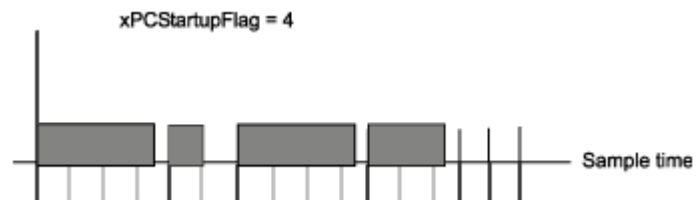
The three properties interact. When the xPC Target kernel runs the model, it checks the number of CPU overloads against the values of `xPCMaxOverloads` and `xPCMaxOverloadLen`. When the number of CPU overloads reaches the lower of these two values, the kernel stops executing the model.

Suppose you enter a line like the following for the **TLC options** parameter:

```
-axPCMaxOverloads=30 -axPCOverLoadLen=2 -axPCStartupFlag=5
```

The software ignores CPU overloads for the first five iterations through the model. After this, the software allows up to 30 CPU overloads, allowing at most two consecutive CPU overloads.

With the TLC options, you can use the following blocks in your model to monitor CPU overloads.

- Use the xPC Target Get Overload Counter and xPC Target Set Overload Counter blocks to set and keep track of CPU overload numbers.

- Use the Pentium Time Stamp Counter block to profile your model.

Getting MathWorks Support

Where Is the MathWorks Support Web Site?

For xPC Target solutions and guidelines, see the following MathWorks Web site resources:

- MathWorks Support xPC Target Web site
  (`http://www.mathworks.com/support/search_results.html?q=product:"xPC+Targ`

  The xPC Target documentation is also available from this site.

- MATLAB Central File Exchange
  (`http://www.mathworks.com/matlabcentral/fileexchange/?term=xPC+Target`)

How Do I Get a Software Update?

1 Navigate to the MathWorks download page (`http://www.mathworks.com/downloads/`).

2 Navigate to the page for the xPC Target software version you want and download it to your host computer.

3 Install and integrate the new release software.

4 Recreate your xPC Target environment. (See "What Should I Do After Updating Software?" on page 25-4.)

What Should I Do After Updating Software?

If you are working with a new xPC Target release, either downloaded from the MathWorks download page (http://www.mathworks.com/downloads/) or installed from a DVD, you must do the following:

1 Type xpcexplr in the MATLAB Command Window.

2 In the **Targets** pane, expand the target computer node.

3 Click the Target Properties icon in the toolbar or double-click **Properties**.

4 Select **Host-to-Target communication** and recreate your xPC Target environment (see "RS-232 Communication Setup" or Network Communication).

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

5 Select **Boot configuration** and click **Create boot disk**.

6 Reboot the target computer.

7 In the Simulink Editor window and from the **Code** menu, click **C/C++ Code > Build Model** for each model to be executed.

How Do I Contact MathWorks Technical Support?

1 If you cannot solve your problem, call function `getxpcinfo` to retrieve diagnostic information for your xPC Target configuration. This function writes the diagnostic information to the file `xpcinfo.txt` in the current folder.

Note The `xpcinfo.txt` file might contain information sensitive to your organization. Review the contents of this file before disclosing it to MathWorks.

2 Contact MathWorks directly for online or phone support:
`http://www.mathworks.com/support/contact_us`

26

Tuning Performance

Building Referenced Models in Parallel

The xPC Target software allows you to build referenced models in parallel on a compute cluster. In this way, you can more quickly build and download xPC Target applications to the target computer.

Note The following procedure assumes you have a functioning xPC Target installation on your host computer.

---

1 Identify a set of worker computers, which might be separate cores on your host computer or computers in a remote cluster running under Windows.

2 If you intend to use separate cores on the host computer, install Parallel Computing Toolbox™ on the host computer.

3 If you intend to use computers in a remote cluster:

  a Install the following on each cluster computer:

    • MATLAB

    • Parallel Computing Toolbox

    • MATLAB Distributed Computing Server™

    • xPC Target

    • Build compiler

    Tip Install the same compiler and compiler version at the same location as on the host computer.

---

  b Start and configure the remote cluster according to the instructions at http://www.mathworks.com/support/product/DM/installation/ver_current/.

4 Run MATLAB on the host computer.

5 In MATLAB, type `matlabpool` to open a MATLAB pool.

6 Type `pctRunOnAll` to configure the compiler for the remote workers as a group. For example:

```
pctRunOnAll('xpcsetCC(''VisualC'',
    ''C:\Program Files\Microsoft Visual Studio 9.0'')')
```

In this configuration, the host computer and the remote workers have installed Microsoft Visual Studio 9.0 at `C:\Program Files\Microsoft Visual Studio 9.0`.

7 Build and download your model.

See "Reduce Build Time for Referenced Models" for more about increasing the speed of parallel builds.

Multicore Processor Configuration

For better performance on your target computer, you can run multirate target applications on multiple cores. Use this capability if your target computer has a multicore processor and you want to take advantage of it for multirate models. Before you consider enabling this capability, see "Target Computer BIOS Settings" for the effects of BIOS settings.

To build and download multirate models on your multiple core target computer:

1 Type xpcexplr in the MATLAB Command Window.

2 In the **Targets** pane, expand the target computer node.

3 Click the Target Properties icon in the toolbar or double-click **Properties**.

4 Select the **Multicore CPU** check box in the **Target settings** pane.

5 Open your model in Simulink Editor.

6 Add a Rate Transition block to transition between rates.

Note Multirate models must use Rate Transition blocks. If your model uses other blocks for rate transitions, building the model generates an error.

7 Select the **Ensure data integrity during data transfer** check box of the Rate Transition block parameters.

8 Clear the **Ensure deterministic data transfer (maximum delay)** check box of the Rate Transition block parameters. This forces the Rate Transition block to use the most recent data available.

Note Because this box is cleared, the transferred data might differ from run to run.

9 In Simulink Editor, select **View > Model Explorer**.

10 In Simulink Model Explorer, right click in the **Model Hierarchy** pane and select **Configuration > Add configuration for concurrent execution**

11 In the new configuration, select **Solver**.

12 Check **Enable concurrent tasking**.

13 Click **Configure Tasks**.

For more on configuring your model for concurrent execution, see "Design Considerations".

Profiling Target Application Execution

You can profile your target computer to see the execution sequence of your target application and then tune the performance. This process is especially useful if your target application is configured to take advantage of multicore processors on the target computer.

To configure your target computer and model to take advantage of multicore processors, see "Multicore Processor Configuration" on page 26-4.

Profiling your target computer requires these steps:

1 Configure the model to enable the collection of profile data during execution.

2 Display and evaluate the profile data.

Profiling adds a slight increase to the execution time of the target application.

In this section...
"Configuring Your Model to Collect Profile Data During Execution" on page 26-6
"Displaying and Evaluating Profile Data" on page 26-7

Configuring Your Model to Collect Profile Data During Execution

To configure your model to collect profile data during execution:

1 In the Simulink editor window, select **Simulation > Model Configuration Parameters**.

2 Open node **xPC Target options** from node **Code Generation**.

3 Select **Enable profiling**.

4 Type a value for the **Number of events** parameter.

By default, the software logs 5000 events for profiling. You can increase or decrease this number. When the software logs the specified number of

events or the model stops, the software stops collecting the data and writes it to *current_working_folder*\xPCTrace.csv on the target computer.

5 Save your changes.

To try this procedure with a preconfigured model, use:

matlabroot\toolbox\rtw\targets\xpc\xpcdemos\xpcprofdemo

To build, download, and display the profile data, use a profiling script as described in "Displaying and Evaluating Profile Data" on page 26-7.

Displaying and Evaluating Profile Data

To see the profile data that your model collects during execution on the target computer, you run a profiling script.

If you have not yet done so, see "Configuring Your Model to Collect Profile Data During Execution" on page 26-6.

In the MATLAB Command Window, run the script:

matlabroot\toolbox\rtw\targets\xpc\xpcdemos\profile_xpc_demo.m

This script:

1 Builds and downloads the model that you used to collect the profile data. By default, the script uses *matlabroot*\toolbox\rtw\targets\xpc\xpcdemos\xpcprofdemo.

2 Saves the data in xPCTrace.csv on the target computer.

xPCTrace.csv is a raw data file that contains information such as a header, version number, row in which data starts, CPU frequency, and the time of the first event.

3 Transfers xPCTrace.csv from the target computer to the current working folder of the target computer.

4 On the host computer, organizes the raw data into the `profileInfo`
structure and displays the profile data in a MATLAB figure window and an
HTML file.

To try this procedure with a example model, use
`profile_xpc_demo.m`. This example calls the script
matlabroot`\toolbox\rtw\targets\xpc\xpcdemos\profile_xpc_demo.m`.

To evaluate the displays, see "Interpreting Profile Data" on page 26-8.

Interpreting Profile Data
The MATLAB figure window displays a task activity graph with the following
entries:
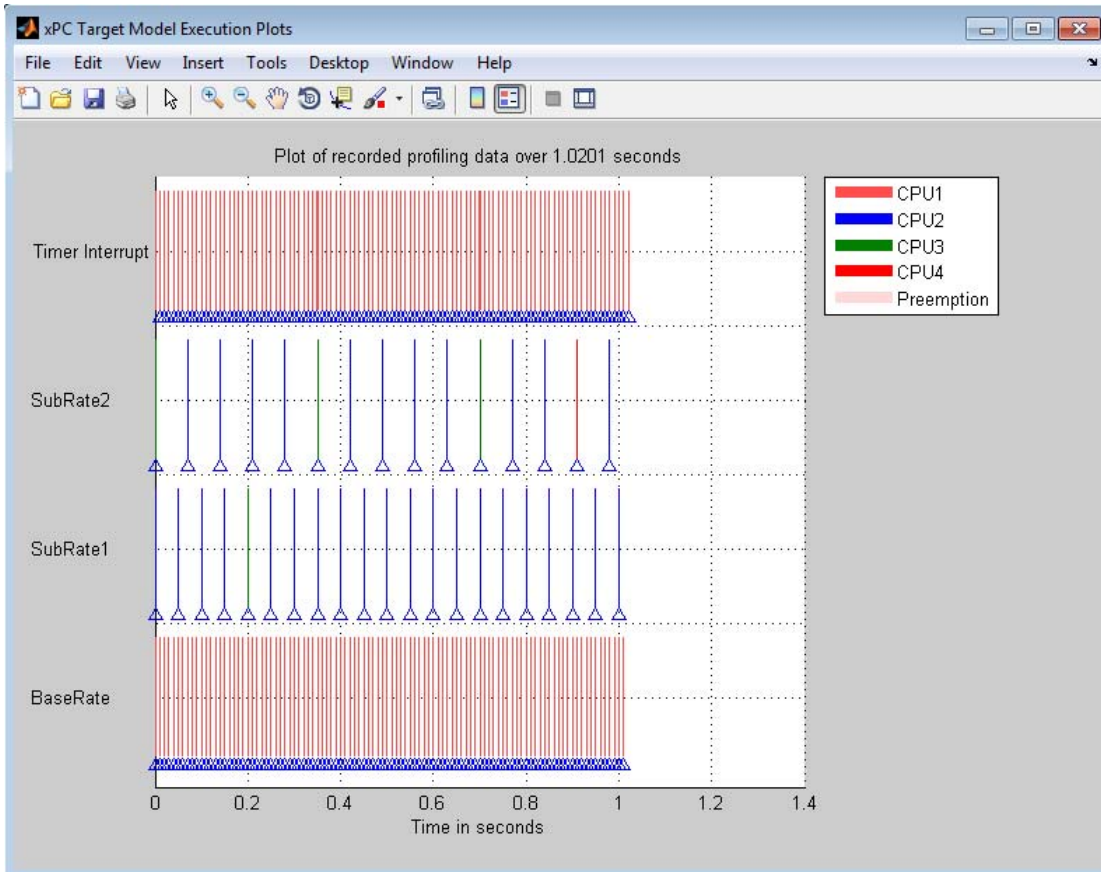
Row	Description
Timer interrupt	Recorded polling data for interrupts
SubRate2	Recorded polling data for SubRate2 task
SubRate1	Recorded polling data for SubRate1 task
BaseRate	Recorded polling data for BaseRate task

The HTML report displays model execution profile results for each task:

Web Browser - xPC Target Report - Execution Profile Results

xPC Target Report - Execution Profile Re... ×   +

Location: C:\Users\ddahlbac\AppData\Local\Temp\xpc_profile_2012_5_31_17_24_34.html

Model Execution Profiling Results

- Analysis of recorded profiling data

All times are in seconds. The CPU frequency is 2.403e+09 Hz, and the timer resolution is 4.1615e-10 seconds.

Analysis of profiling data recorded over 1.0201 seconds.

Profiling data was recorded over 1.0201 seconds. The recorded data for task turnaround times and task execution times is presented in the table below.

Task	Maximum turnaround time	Average turnaround time	Maximum execution time	Average execution time	Average sample time
BaseRate	2.48e-05 at 0.35	1.69e-05	2.48e-05 at 0.35	1.69e-05	0.01
SubRate1	8.28e-06 at 7.13e-06	3.09e-06	8.28e-06 at 7.13e-06	3.09e-06	0.05
SubRate2	7.15e-06 at 8.25e-06	3.16e-06	7.15e-06 at 8.25e-06	3.16e-06	0.07
Timer Interrupt	2.37e-06 at 0.21	1.45e-06	2.37e-06 at 0.21	1.45e-06	0.010001

If there is insufficient data, values show N/A (Not Available).

Task turnaround time is the elapsed time between start and finish of the task. If the task is not preempted, the task turnaround time equals the task execution t

Task execution time is the time between task start and finish, when the task is actually running and not preempted by another task. The task execution time ca
be measured directly; it is inferred from the task start and finish time and the intervening periods during which another task has preempted it. Note that, in perfor
these calculations, processor time consumed by the scheduler while switching tasks is not taken into account. This means that, in cases where preemption occur
the reported task execution times overestimate the true values.

Result	Description
Maximum turnaround time	Longest time between when the task starts and finishes. This time includes task preemptions (interrupts). If no preemptions occur, the maximum turnaround time is the same as the maximum task execution time.
Average turnaround time	Average time between when the task starts and finishes. This time includes task preemptions (interrupts). If no preemptions occur, the average turnaround time is the same as the average task execution time.
Maximum execution time	Longest time between when the task starts and finishes. This time does not include task preemptions (interrupts).
Average execution time	Average time between when the task starts and finishes. This time does not include task preemptions (interrupts).
Average sample time	Average sample time of the task.

Customizing profile_xpc_demo.m

The profiling script, profile_xpc_demo.m, works with the example model:

matlabroot\toolbox\rtw\targets\xpc\xpcdemos\xpcprofdemo

To customize this script:

1 Configure your model as described in "Configuring Your Model to Collect Profile Data During Execution" on page 26-6.

2 Copy
  matlabroot\toolbox\rtw\targets\xpc\xpcdemos\profile_xpc_demo.m
  to your working area and rename the copy. For example, use
  the name my_profile_xpc_demo.m.

3 Edit my_profile_xpc_demo.m.

4 Notice that the my_profile_xpc_demo.m script uses the profileInfo structure, which allows you to customize the script. This structure contains the following fields:

Field	Description	Values
rawdataonhost	Location of the raw data. If this field does not exist, the script assumes a value of 0.	0 — Default. Transfer the raw data from the target computer to the host. 1 — Look for the saved raw data file xPCTrace.csv in the current folder on the host computer.
modelname	Name of the model	xpcprofdemo — Default. The name of a model in the xpcdemos folder. *your_model_name* — Specify the name of your model that you want to profile.
noplot	Specify whether or not to display the model execution plots on the host computer monitor.	0 — Default. Do not display the model execution plots on the host computer monitor. 1 — Display the model execution plots on the host computer monitor.
noreport	Specify whether or not to display the model profiling result report on the host computer monitor.	0 — Default. Do not display the model profiling result report on the host computer monitor. 1 — Display the model profiling result report on the host computer monitor.

5 Change values as desired. For example, to profile your own model, replace instances of `xpcprofdemo` with your model name, such as `my_xpcprofdemo`.

6 Run your custom script.

Functions

fc422mexcalcbits

Purpose   Calculate parameter values for Fastcom 422/2-PCI board

Syntax   `[a b] = fc422mexcalcbits(frequency)`
`[a b df] = fc422mexcalcbits(frequency)`

Arguments

`frequency`	Desired baud rate for the board

`[a b] = fc422mexcalcbits(frequency)` accepts a baud rate (in units of baud/second) and converts this value into two parameters `a b`. You must enter these values for the parameter **Clock bits** of the Fastcom® 422/2-PCI driver clock. The desired baud rate (`frequency`) must range between 30e3 and 1.5e6, which is a hardware limitation of the clock circuit.

`[a b df] = fc422mexcalcbits(frequency)` accepts a baud rate (in units of baud/second) and converts this value into two parameters `a b`. You must enter these values for the parameter **Clock bits** of the Fastcom 422/2-PCI driver block. The third value, `df`, indicates the actual baud rate that is created by the generated parameters `a b`. The clock circuit has limited resolution and is unable to perfectly match an arbitrary frequency. The desired baud rate (`frequency`) must range between 30e3 and 1.5e6, which is a hardware limitation of the clock circuit.

Purpose   List environment properties assigned to MATLAB variable

Syntax   getxpcenv
                 getxpcenv propertyname

Description getxpcenv displays, in the MATLAB Command Window, the property
                 names and current property values for the xPC Target environment.

                 getxpcenv propertyname displays the current value of property
                 propertyname. The environment properties define communication
                 between the host computer and target computer and the type of target
                 boot kernel created during the setup process.

                 Tip To access a subset of these properties in xPC Target Explorer:

                 1 Expand a target computer node in the **Targets** pane.

                 2 Click the Target Properties icon ![icon] in the toolbar or double-click
                 Properties.

                 - "Host-to-Target Communication" on page 27-4
                 - "Target Settings" on page 27-10
                 - "Boot Configuration" on page 27-13
                 - "Host Configuration" on page 27-16

Host-to-Target Communication

Environment Property	Description
HostTargetComm	Property values are `'RS232'` and `'TcpIp'`.
	Select `RS-232` or `TCP/IP` from the **Communication type** list in the **Target Properties** pane of xPC Target Explorer.
	If you select `RS-232`, you also must set the property `RS232HostPort`. If you select `TCP/IP`, then you also need to set all properties that start with `TcpIp`.
	Note RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.
RS232Baudrate	Property values are `'115200'`, `'57600'`, `'38400'`, `'19200'`, `'9600'`, `'4800'`, `'2400'`, and `'1200'`.
	Select 1200, 2400, 48 00, 9600, 19200, 38400, 57600, or 115200 from the **Baud rate** list in the **Target Properties** pane of xPC Target Explorer.

Environment Property	Description
RS232HostPort	Property values are `'COM1'` and `'COM2'`.
	Select `COM1` or `COM2` from the **Host port** list in the **Target Properties** pane of xPC Target Explorer. The software automatically determines the COM port on the target computer.
	Before you can select an RS-232 port, you need to set the `HostTargetComm` property to `RS232`.
TcpIpGateway	Property value is `'xxx.xxx.xxx.xxx'`.
	Enter the IP address for your gateway in the **Gateway** box in the **Target Properties** pane of xPC Target Explorer. This property is set by default to `255.255.255.255`, which means that a gateway is not used to connect to the target computer.
	If you communicate with your target computer from within a LAN that uses gateways, and your host and target computers are connected through a gateway, you must 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.

Environment Property	Description
TcpIpSubNetMask	Property value is '`xxx.xxx.xxx.xxx`'.
	Enter the subnet mask of your LAN in the **Subnet mask** box in the **Target Properties** pane of xPC Target Explorer. Ask your system administrator for this value.
	For example, your subnet mask could be `255.255.255.0`.
TcpIpTargetAddress	Property value is '`xxx.xxx.xxx.xxx`'.
	Enter a valid IP address for your target computer in the **IP address** box in the **Target Properties** pane of xPC Target Explorer. Ask your system administrator for this value.
	For example, `192.168.0.10`.
TcpIpTargetBusType	Property values are '`PCI`', '`ISA`', and '`USB`'.
	Select `PCI`, `ISA`, or `USB` from the **Bus type** list in the **Target Properties** pane of xPC Target Explorer. This property is set by default to `PCI`, and determines the bus type of your target computer. You do not need to define a bus type for your host computer, which can be the same or different from the bus type in your target computer.

Environment Property	Description
	If `TcpIpTargetBusType` is set to `PCI`, then the properties `TcpIpISAMemPort` and `TcpIpISAIRQ` have no effect on TCP/IP communication.
	If you are using an ISA bus card, set `TcpIpTargetBusType` to `ISA` and enter values for `TcpIpISAMemPort` and `TcpIpISAIRQ`.
`TcpIpTargetDriver`	Property values are `'3C90x'`, `'I8254x'`, `'I82559'`, `'NE2000'`, `'NS83815'`, `'R8139'`, `'R8168'`, `'Rhine'`, `'RTLANCE'`, `'SMC91C9X'`, `'USBAX772'`, `'USBAX172'`, and `'Auto'`.
	Select `THREECOM_3C90x`, `INTEL_I8254x`, `INTEL_I82559`, `NE2000`, `NS83815`, `R8139`, `R8168`, `Rhine`, `RTLANCE`, `SMC91C9X`, `USBAX772`, `USBAX172`, or `Auto` from the **Target driver** list in the **Target Properties** pane of xPC Target Explorer.

Environment Property	Description
TcpIpTargetISAIRQ	Property value is 'n', where *n* is between 5 and 15 inclusive.
	Select an IRQ value from the **IRQ** list in the **Target Properties** pane of xPC Target Explorer.
	If you are using an ISA bus Ethernet card, you must enter values for the properties TcpIpISAMemPort and TcpIpISAIRQ. The values of these properties must correspond to the jumper settings or ROM settings on the ISA-bus Ethernet card.
	On your ISA bus card, assign an IRQ and I/O-port base address by moving the jumpers on the card.
	Set the IRQ to 5, 10, or 11. If one of these hardware settings leads to a conflict in your target computer, choose another IRQ and make the corresponding changes to your jumper settings.

Environment Property	Description
`TcpIpTargetISAMemPort`	Property value is `'0xnnnn'`.
	Enter an I/O port base address in the **Address** box in the **Target Properties** pane of xPC Target Explorer.
	If you are using an ISA bus Ethernet card, you must enter values for the properties `TcpIpISAMemPort` and `TcpIpISAIRQ`. The values of these properties must correspond to the jumper settings or ROM settings on your ISA bus Ethernet card.
	On your ISA bus card, assign an IRQ and I/O port base address by moving the jumpers on the card.
	Set the I/O port base address to around 0x300. If one of these hardware settings leads to a conflict in your target computer, choose another I/O port base address and make the corresponding changes to your jumper settings.
`TcpIpTargetPort`	Property value is `'xxxxx'`.
	Enter a port address greater than 20000 in the **Port** box in the **Target Properties** pane of xPC Target Explorer.
	This property is set by default to 22222. The default value is higher than the reserved area

Environment Property	Description
	(telnet, ftp, . . .) and is only of use on the target computer.

Target Settings

Environment Property	Description
EthernetIndex	Property value is 'n', where *n* indicates the index number for the Ethernet card on a target computer. Note that the (n-1)th Ethernet card on the target computer has an index number 'n'. The default index number is 0.
	If the target computer has multiple Ethernet cards, you must select one of the cards for host-target communication. This option returns the index number of the card selected on the target computer upon booting.
LegacyMultiCoreConfig	Property values are 'on' and 'off' (the default).
	Set this value to 'on' only if your target computer contains hardware not compliant with the Advanced Configuration and Power Interface (ACPI) standard. Otherwise, set this value to 'off'.
MaxModelSize	Property values are '1MB' (the default), '4MB', and '16MB'.
	Select 1 MB, 4 MB, or 16 MB from the **Model size** list in the **Target Properties** pane of xPC Target Explorer.
	Choosing the maximum model size reserves the specified amount of memory on the target computer for the target application. Memory not used by the target application is used by the kernel and by the heap for data logging.

Environment Property	Description
	Selecting too high a value leaves less memory for data logging. Selecting too low a value does not reserve enough memory for the target application and creates an error. You can approximate the size of the target application by the size of the DLM file produced by the build process. • BootFloppy and DOSLoader modes ignore this value. • In StandAlone mode, you can only use MaxModelSize values '1MB' and '4MB'.
MulticoreSupport	Property values are 'on' and 'off' (the default). Select or clear the **Multicore CPU** check box in the **Target Properties** pane of xPC Target Explorer. If your target computer has multicore processors, set this value to 'on' to take advantage of these processors for background tasks. Otherwise, set this value to 'off'.
Name	Target computer name.
NonPentiumSupport	Property values are 'on' and 'off' (the default). Select or clear the **Target is a 386/486** check box in the **Target Properties** pane of xPC Target Explorer. Set this value to 'on' if your target computer has a 386 or 486 compatible processor. Otherwise, set it to 'off'. If your target computer has a Pentium or higher compatible processor, selecting this check box slows the performance of your target computer.

Environment Property	Description
SecondaryIDE	Property values are 'on' and 'off' (the default).
	Select or clear the **Secondary IDE** check box in the **Target Properties** pane of xPC Target Explorer.
	Set this value to 'on' only if you want to use the disks connected to a secondary IDE controller. If you do not have disks connected to the secondary IDE controller, leave this value set to 'off'.
ShowHardware	Property values are 'on' and 'off' (the default).
	If you create a target boot kernel when ShowHardware is 'on' and boot the target computer with it, the kernel displays the index, bus, slot, function, and target driver for each Ethernet card on the target monitor.
	The host computer cannot communicate with the target computer after the kernel boots with ShowHardware set.
TargetRAMSizeMB	Property values are 'Auto' (the default) and 'xxx', where *xxx* is a positive value specifying the amount of RAM, in megabytes, installed on the target computer.
	Under **RAM size**, click the **Auto** or **Manual** button in the **Target Properties** pane of xPC Target Explorer. If you click **Manual**, enter the amount of RAM, in megabytes, installed on the target computer in the **Size(MB)** box.
	TargetRAMSizeMB defines the total amount of installed RAM in the target computer. This RAM is used for the kernel, target application, data logging, and other functions that use the heap.
	If TargetRAMSizeMB is assigned 'Auto', the target application reads the target computer BIOS and determines the amount of memory up to a maximum of 2 GB. If the target application cannot read the BIOS,

Environment Property	Description
	you must select Manual mode and enter the amount of memory, in megabytes, up to a maximum of 2 GB.
	The xPC Target kernel can use only 2 GB of memory.
TargetScope	Property values are 'Disabled' and 'Enabled' (the default).
	Select or clear the **Graphics mode** check box in the **Target Properties** pane of xPC Target Explorer.
	If you set TargetScope to Disabled, the target computer displays information as text.
	To use the full features of a target scope, install a keyboard on the target computer.
USBSupport	Property values are 'on' (the default) and 'off'.
	Select or clear the **USB Support** check box in the **Target Properties** pane of xPC Target Explorer.
	Set this value to 'on' if you want to use a USB port on the target computer; for example, to connect a USB mouse. Otherwise, set it to 'off'.

Boot Configuration

Environment Property	Description
BootFloppyLocation	Drive name for creation of target boot disk.
CDBootImageLocation	Location of cdboot.iso file for creation of CD target boot disk.
DOSLoaderLocation	Location of DOSLoader files to boot target computers from devices other than floppy disk or CD.

Environment Property	Description
EmbeddedOption	Property values are `'Disabled'` and `'Enabled'`. This property is read only.
	Note that the xPC Target Embedded Option product is enabled only if you purchase an additional license.
TargetBoot	Property values are `'BootFloppy'`, `'CDBoot'`, `'DOSLoader'`, `'NetworkBoot'`, and `'StandAlone'`.
	Select Removable Disk, CD, DOS Loader, Network, or Stand Alone from the **Boot mode** list in the **Target Properties** pane of xPC Target Explorer.
	If your license file does not include the license for the xPC Target Embedded Option product, your only options are Removable Disk, CD, DOS Loader, and Network. With the xPC Target Embedded Option product licensed and installed, you have the additional choice of Stand Alone.

Environment Property	Description
	Tip In the **Target Properties** pane of xPC Target Explorer, click the **Create boot disk** button to create a bootable image in the specified boot mode.
TargetMACAddress	Physical target computer MAC address from which to accept boot requests when booting within a dedicated network. Format the MAC address as six pairs of hexadecimal numbers, separated by colons: xx:xx:xx:xx:xx:xx To update the MAC address in xPC Target Explorer, first click the **Reset** button in the **Target Properties** pane. You can then click the **Specify new MAC address** button to enter a MAC address manually in the **MAC address** box. If you do not enter a MAC address manually, the software will obtain the MAC address automatically the next time you restart the target computer.

getxpcenv

Host Configuration

Environment Property	Description
Version	xPC Target version number. Displayed only from getxpcenv when called without arguments.

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

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

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

```
env = getxpcenv
```

See Also   setxpcenv | xpcbootdisk

Purpose
Retrieve diagnostic information to help troubleshoot configuration issues

Syntax
```
getxpcinfo
getxpcinfo('-a')
```

Arguments

`'-a'`	Appends diagnostic information to an existing xpcinfo.txt file. If one does not exist, this function creates the file in the current folder.

Description
getxpcinfo returns diagnostic information for troubleshooting xPC Target configuration issues. This function generates and saves the information in the xpcinfo.txt file, in the current folder. If the file xpcinfo.txt already exists, this function overwrites it with the new information.

getxpcinfo('-a') appends the diagnostic information to the xpcinfo.txt file, in the current folder. If the file xpcinfo.txt does not exist, this function creates it.

You can send the file xpcinfo.txt to MathWorks Technical Support for evaluation and guidance. To create this file, you must have write permission for the current folder.

Warning

The file xpcinfo.txt might contain information sensitive to your organization. Review the contents of this file before sending to MathWorks.

macaddr

Purpose	Convert string-based MAC address to vector-based one

Syntax

```
macaddr('MAC address')
```

Argument

'MAC address'   String-based MAC address to be converted.

Description

macaddr('MAC address') converts a string-based MAC address to a vector-based MAC address. The string-based MAC address should be a string comprised of six colon-delimited fields of two-digit hexadecimal numbers.

Examples

```
macaddr('01:23:45:67:89:ab')

ans =

    1    35    69   103   137   171
```

How To

- "Model-Based Ethernet Communications"

Purpose   Change xPC Target environment properties

Syntax   setxpcenv
                   setxpcenv('property_name','property_value')
                   setxpcenv('prop_name1','prop_value1','prop_name2',. . .)

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. setxpcenv works similarly to the set function of the MATLAB Handle Graphics® system.

setxpcenv called without arguments returns a list of allowed property values in the MATLAB window.

setxpcenv('property_name','property_value') sets property property_name to property_value.

setxpcenv('prop_name1','prop_value1','prop_name2',. . .) is called with one or more argument pairs. The first argument of a pair is the property name; the second is the new value for this property.

The environment properties define communication between the host computer and target computer and the type of target boot kernel created during the setup process. With the exception of the Version property, you can set environment properties using the setxpcenv function or the xPC Target Explorer window, accessed via the xpcexplr function. An understanding of the environment properties will help you configure the xPC Target environment.

Tip To access a subset of these properties in xPC Target Explorer:

1 Expand a target computer node in the **Targets** pane.

2 Click the Target Properties icon in the toolbar or double-click **Properties**.

- "Host-to-Target Communication" on page 27-20
- "Target Settings" on page 27-26
- "Boot Configuration" on page 27-30

Host-to-Target Communication

Environment Property	Description
HostTargetComm	Property values are 'RS232' and 'TcpIp'.
	Select RS-232 or TCP/IP from the **Communication type** list in the **Target Properties** pane of xPC Target Explorer.
	If you select RS-232, you also must set the property RS232HostPort. If you select TCP/IP, then you also need to set all properties that start with TcpIp.

Environment Property	Description
	Note RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.
RS232Baudrate	Property values are '115200', '57600', '38400', '19200', '9600', '4800', '2400', and '1200'.
	Select 1200, 2400, 48 00, 9600, 19200, 38400, 57600, or 115200 from the **Baud rate** list in the **Target Properties** pane of xPC Target Explorer.
RS232HostPort	Property values are 'COM1' and 'COM2'.
	Select COM1 or COM2 from the **Host port** list in the **Target Properties** pane of xPC Target Explorer. The software automatically determines the COM port on the target computer.
	Before you can select an RS-232 port, you need to set the HostTargetComm property to RS232.

Environment Property	Description
`TcpIpGateway`	Property value is `'xxx.xxx.xxx.xxx'`.
	Enter the IP address for your gateway in the **Gateway** box in the **Target Properties** pane of xPC Target Explorer. This property is set by default to `255.255.255.255`, which means that a gateway is not used to connect to the target computer.
	If you communicate with your target computer from within a LAN that uses gateways, and your host and target computers are connected through a gateway, you must 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.
`TcpIpSubNetMask`	Property value is `'xxx.xxx.xxx.xxx'`.
	Enter the subnet mask of your LAN in the **Subnet mask** box in the **Target Properties** pane of xPC Target Explorer. Ask your system administrator for this value.
	For example, your subnet mask could be `255.255.255.0`.

Environment Property	Description
TcpIpTargetAddress	Property value is `'xxx.xxx.xxx.xxx'`.
	Enter a valid IP address for your target computer in the **IP address** box in the **Target Properties** pane of xPC Target Explorer. Ask your system administrator for this value.
	For example, `192.168.0.10`.
TcpIpTargetBusType	Property values are `'PCI'`, `'ISA'`, and `'USB'`.
	Select `PCI`, `ISA`, or `USB` from the **Bus type** list in the **Target Properties** pane of xPC Target Explorer. This property is set by default to `PCI`, and determines the bus type of your target computer. You do not need to define a bus type for your host computer, which can be the same or different from the bus type in your target computer.
	If `TcpIpTargetBusType` is set to `PCI`, then the properties `TcpIpISAMemPort` and `TcpIpISAIRQ` have no effect on TCP/IP communication.
	If you are using an ISA bus card, set `TcpIpTargetBusType` to `ISA` and enter values for `TcpIpISAMemPort` and `TcpIpISAIRQ`.

Environment Property	Description
TcpIpTargetDriver	Property values are '3C90x', 'I8254x', 'I82559', 'NE2000', 'NS83815', 'R8139', 'R8168', 'Rhine', 'RTLANCE', 'SMC91C9X', 'USBAX772', 'USBAX172', and 'Auto'.
	Select THREECOM_3C90x, INTEL_I8254x, INTEL_I82559, NE2000, NS83815, R8139, R8168, Rhine, RTLANCE, SMC91C9X, USBAX772, USBAX172, or Auto from the **Target driver** list in the **Target Properties** pane of xPC Target Explorer.
TcpIpTargetISAIRQ	Property value is 'n', where *n* is between 5 and 15 inclusive.
	Select an IRQ value from the **IRQ** list in the **Target Properties** pane of xPC Target Explorer.
	If you are using an ISA bus Ethernet card, you must enter values for the properties TcpIpISAMemPort and TcpIpISAIRQ. The values of these properties must correspond to the jumper settings or ROM settings on the ISA-bus Ethernet card.
	On your ISA bus card, assign an IRQ and I/O-port base address by moving the jumpers on the card.
	Set the IRQ to 5, 10, or 11. If one of these hardware settings

Environment Property	Description
	leads to a conflict in your target computer, choose another IRQ and make the corresponding changes to your jumper settings.
TcpIpTargetISAMemPort	Property value is '0xnnnn'.
	Enter an I/O port base address in the **Address** box in the **Target Properties** pane of xPC Target Explorer.
	If you are using an ISA bus Ethernet card, you must enter values for the properties TcpIpISAMemPort and TcpIpISAIRQ. The values of these properties must correspond to the jumper settings or ROM settings on your ISA bus Ethernet card.
	On your ISA bus card, assign an IRQ and I/O port base address by moving the jumpers on the card.
	Set the I/O port base address to around 0x300. If one of these hardware settings leads to a conflict in your target computer, choose another I/O port base address and make the corresponding changes to your jumper settings.
TcpIpTargetPort	Property value is 'xxxxx'.
	Enter a port address greater than 20000 in the **Port** box in the

Environment Property	Description
	Target Properties pane of xPC Target Explorer.
	This property is set by default to 22222. The default value is higher than the reserved area (telnet, ftp, . . .) and is only of use on the target computer.

Target Settings

Environment Property	Description
EthernetIndex	Property value is 'n', where *n* indicates the index number for the Ethernet card on a target computer. Note that the (n-1)th Ethernet card on the target computer has an index number 'n'. The default index number is 0.
	If the target computer has multiple Ethernet cards, you must select one of the cards for host-target communication. This option returns the index number of the card selected on the target computer upon booting.
LegacyMultiCoreConfig	Property values are 'on' and 'off' (the default).
	Set this value to 'on' only if your target computer contains hardware not compliant with the Advanced Configuration and Power Interface (ACPI) standard. Otherwise, set this value to 'off'.

Environment Property	Description
MaxModelSize	Property values are '1MB' (the default), '4MB', and '16MB'.
	Select 1 MB, 4 MB, or 16 MB from the **Model size** list in the **Target Properties** pane of xPC Target Explorer.
	Choosing the maximum model size reserves the specified amount of memory on the target computer for the target application. Memory not used by the target application is used by the kernel and by the heap for data logging.
	Selecting too high a value leaves less memory for data logging. Selecting too low a value does not reserve enough memory for the target application and creates an error. You can approximate the size of the target application by the size of the DLM file produced by the build process. • BootFloppy and DOSLoader modes ignore this value.
	• In StandAlone mode, you can only use MaxModelSize values '1MB' and '4MB'.
MulticoreSupport	Property values are 'on' and 'off' (the default).
	Select or clear the **Multicore CPU** check box in the **Target Properties** pane of xPC Target Explorer.
	If your target computer has multicore processors, set this value to 'on' to take advantage of these processors for background tasks. Otherwise, set this value to 'off'.
Name	Target computer name.

Environment Property	Description
NonPentiumSupport	Property values are `'on'` and `'off'` (the default).
	Select or clear the **Target is a 386/486** check box in the **Target Properties** pane of xPC Target Explorer.
	Set this value to `'on'` if your target computer has a 386 or 486 compatible processor. Otherwise, set it to `'off'`. If your target computer has a Pentium or higher compatible processor, selecting this check box slows the performance of your target computer.
SecondaryIDE	Property values are `'on'` and `'off'` (the default).
	Select or clear the **Secondary IDE** check box in the **Target Properties** pane of xPC Target Explorer.
	Set this value to `'on'` only if you want to use the disks connected to a secondary IDE controller. If you do not have disks connected to the secondary IDE controller, leave this value set to `'off'`.
ShowHardware	Property values are `'on'` and `'off'` (the default).
	If you create a target boot kernel when ShowHardware is `'on'` and boot the target computer with it, the kernel displays the index, bus, slot, function, and target driver for each Ethernet card on the target monitor.
	The host computer cannot communicate with the target computer after the kernel boots with ShowHardware set.

Environment Property	Description
TargetRAMSizeMB	Property values are `'Auto'` (the default) and `'xxx'`, where *xxx* is a positive value specifying the amount of RAM, in megabytes, installed on the target computer.
	Under **RAM size**, click the **Auto** or **Manual** button in the **Target Properties** pane of xPC Target Explorer. If you click **Manual**, enter the amount of RAM, in megabytes, installed on the target computer in the **Size(MB)** box.
	TargetRAMSizeMB defines the total amount of installed RAM in the target computer. This RAM is used for the kernel, target application, data logging, and other functions that use the heap.
	If TargetRAMSizeMB is assigned `'Auto'`, the target application reads the target computer BIOS and determines the amount of memory up to a maximum of 2 GB. If the target application cannot read the BIOS, you must select Manual mode and enter the amount of memory, in megabytes, up to a maximum of 2 GB.
	The xPC Target kernel can use only 2 GB of memory.
TargetScope	Property values are `'Disabled'` and `'Enabled'` (the default).
	Select or clear the **Graphics mode** check box in the **Target Properties** pane of xPC Target Explorer.
	If you set TargetScope to Disabled, the target computer displays information as text.
	To use the full features of a target scope, install a keyboard on the target computer.
USBSupport	Property values are `'on'` (the default) and `'off'`.
	Select or clear the **USB Support** check box in the **Target Properties** pane of xPC Target Explorer.

Environment Property	Description
	Set this value to `'on'` if you want to use a USB port on the target computer; for example, to connect a USB mouse. Otherwise, set it to `'off'`.

Boot Configuration

Environment Property	Description
BootFloppyLocation	Drive name for creation of target boot disk.
CDBootImageLocation	Location of cdboot.iso file for creation of CD target boot disk.
DOSLoaderLocation	Location of DOSLoader files to boot target computers from devices other than floppy disk or CD.
EmbeddedOption	Property values are `'Disabled'` and `'Enabled'`. This property is read only. Note that the xPC Target Embedded Option product is enabled only if you purchase an additional license.

Environment Property	Description
TargetBoot	Property values are `'BootFloppy'`, `'CDBoot'`, `'DOSLoader'`, `'NetworkBoot'`, and `'StandAlone'`.
	Select `Removable Disk`, `CD`, `DOS Loader`, `Network`, or `Stand Alone` from the **Boot mode** list in the **Target Properties** pane of xPC Target Explorer.
	If your license file does not include the license for the xPC Target Embedded Option product, your only options are `Removable Disk`, `CD`, `DOS Loader`, and `Network`. With the xPC Target Embedded Option product licensed and installed, you have the additional choice of `Stand Alone`.
	Tip In the **Target Properties** pane of xPC Target Explorer, click the **Create boot disk** button to create a bootable image in the specified boot mode.
TargetMACAddress	Physical target computer MAC address from which to accept boot requests when booting within a dedicated network. Format the MAC address as six pairs of hexadecimal numbers, separated by colons:

Environment Property	Description
	xx:xx:xx:xx:xx:xx To update the MAC address in xPC Target Explorer, first click the **Reset** button in the **Target Properties** pane. You can then click the **Specify new MAC address** button to enter a MAC address manually in the **MAC address** box. If you do not enter a MAC address manually, the software will obtain the MAC address automatically the next time you restart the target computer.

Examples   List the current environment properties.

setxpcenv

Change the serial communication port of the host computer to COM2.

setxpcenv('RS232HostPort','COM2')

See Also   getxpcenv | xpcbootdisk

How To
- "Ethernet Communication Setup"
- "RS-232 Communication Setup"
- "Target Boot Methods"
- "Command-Line Setup" on page 4-9

Purpose   Benchmark xPC Target models on target computer

Syntax
```
xpcbench
xpcbench benchmark
xpcbench benchmark -reboot
xpcbench benchmark -cleanup
xpcbench benchmark -verbose
xpcbench benchmark -reboot -cleanup -verbose

expected_results = xpcbench()
current_results = xpcbench(benchmark, ___ )
```

Description   xpcbench benchmarks the real-time execution performance of xPC
Target applications on your target computer. It compares the result to
stored benchmark results from other computers.

Benchmark execution includes generating benchmark models, building
and downloading xPC Target applications, searching for the minimal
achievable sample time, and displaying results.

xpcbench without an argument displays two plots containing
representative results for five benchmarks run on various target
computers.

- For each target computer tested, the first plot lists the minimal
  achievable sample time for the five benchmarks, in microseconds.

- The second plot contains a bar graph of all computers tested, ranked
  by relative performance.

The results are labeled by CPU type, CPU clock rate, and the compiler
for the application.

Depending upon the value of benchmark, xpcbench benchmark produces
different outputs:

xpcbench

- `xpcbench this` displays two plots containing benchmark results for the five benchmarks run on your target computer, compared with the representative benchmark results for other target computers:

  - For each target computer tested, the first plot lists the minimal achievable sample time for the benchmarks, in microseconds.

  - The second plot contains a bar graph of all computers tested, ranked by relative performance.

  The results are labeled by CPU type, CPU clock rate, and the compiler used to compile the application. The entry for your target computer is highlighted.

- `xpcbench benchmark` prints the benchmark name, the number of blocks, the model build time in seconds, the execution time in seconds, and the minimal achievable sample time in microseconds in the MATLAB command window.

`xpcbench benchmark -reboot` runs the benchmark, then restarts the target computer.

`xpcbench benchmark -cleanup` runs the benchmark, plots or prints benchmark results, and deletes the build files.

`xpcbench benchmark -verbose` prints build output, runs the benchmark, and plots or prints benchmark results.

`xpcbench benchmark -reboot -cleanup -verbose` prints build output, restarts the target computer, deletes build files, and plots or prints results.

You can add zero or more of these control arguments in arbitrary order.

`expected_results = xpcbench()` returns the benchmark results for the five predefined benchmarks in a structure array.

Depending upon the value of benchmark, current_results = xpcbench(benchmark, ___) returns different results:

- xpcbench('this') returns the benchmark results for the five predefined benchmarks in a structure array.

- xpcbench(benchmark) returns the benchmark results for the specified model in a structure.

Input Arguments

benchmark - Benchmark name or model name

this | *usermdl* | minimal | f14 | f14*5 | f14*10 | f14*25

Benchmark, specified as a literal string or string variable containing one of:

this	All five predefined benchmark models (minimal, f14, f14*5, f14*10, f14*25)
usermdl	Your model, *usermdl*.
minimal	Minimal model consisting of three blocks (Constant, Gain, Termination).
f14	Standard Simulink example f14 (62 blocks, 10 continuous states).
f14*5	Five f14 systems modeled in subsystems (310 blocks, 50 continuous states).
f14*10	Ten f14 systems (620 blocks, 100 continuous states).
f14*25	25 f14 systems (1550 blocks, 250 continuous states).

When using function form, enclose literal arguments (this, -reboot) in single quotes ('this','-reboot').

Example:

Data Types
char

Output Arguments

expected_results - Results of predefined benchmarks previously run on representative target computers

struct array

xpcbench without an argument returns representative benchmark results in a structure array with element fields:

Machine	Target computer information string containing CPU type, CPU speed, compiler
BenchResults	Target computer benchmark performance for all five predefined benchmarks
Desc	Target computer descriptor string containing machine type, RAM size, cache size

current_results - Current results of specified benchmark

struct

xpcbench(benchmark) returns the benchmark results in a structure with fields:

Name	Benchmark name
nBlocks	Number of blocks in benchmark
BuildTime	Elapsed time in seconds to build benchmark
BenchTime	Elapsed time in seconds to run benchmark
Tsmin	Minimal achievable sample time in seconds for benchmark

Tips
- Before you run `xpcbench`, you must be able to start the target computer, connect the host computer to the target computer, and run the confidence test, `xpctest`, with no failures.

- After running `xpcbench` on your model and system, set your model sample time to the minimal achievable sample time value reported. Smaller sample times overload the target computer.

- The stored benchmark results were collected with **Multicore CPU support** disabled. When evaluating your system, temporarily disable this target setting using `xpcexplr`.

- The stored benchmark models were compiled using a sampling of the currently supported compilers. When evaluating your system, find the closest match to the compiler you are using.

- Benchmark `minimal` has neither continuous nor discrete states. It provides information about the target computer interrupt latencies.

Examples

xpcbench

Show representative benchmark results from various target computers.

Start the target computer and run confidence test.

```
xpcbench_prework_snippet
```

Display representative results on five predefined benchmarks.

```
xpcbench
```

xpcbench

```
xPC Target Benchmark - Details

File  Edit  View  Insert  Tools  Desktop  Window  Help

          Minimal achievable sample times in μ s

                                                    Minimal   F14   F14*5   F14*10   F14*25
Intel Core 2 Duo E6420   2.13GHz (VisualC9.0)          8       9      11      14      21
Intel Core 2 Duo E6420   2.13GHz (VisualC10.0)         8      10      11      13      22
AMD Athlon X2 4600+       2.4GHz  (VisualC9.0)          8       9      11      14      25
AMD Athlon X2 4600+       2.4GHz  (VisualC10.0)         8       9      12      14      25
Intel Quad-Core Q6600    2.48Ghz (VisualC9.0)          8      12      13      16      30
Intel Quad-Core Q6600    2.48Ghz (VisualC10.0)         8      10      13      17      30
Intel Pentium M          1.8GHz  (VisualC10.0)         8       8      13      18      36
Intel Pentium M          1.8GHz  (VisualC9.0)          8      10      13      17      36
Intel Core 2 Duo E6420   2.13GHz (VisualC6.0)          8      10      13      19      33
Intel Core 2 Duo E6420   2.13GHz (OpenWatcom1.8)       8       9      14      19      33
Intel Pentium 4          2.8GHz  (VisualC9.0)          8      10      14      17      37
Intel Pentium 4          2.8GHz  (VisualC10.0)         8      10      13      18      38
Intel Core 2 Duo         1.5GHz  (VisualC9.0)          9       9      18      23      43
Intel Core 2 Duo         1.5GHz  (VisualC10.0)         9      12      17      24      44
Intel Quad-Core Q6600    2.48Ghz (VisualC6.0)          8      13      18      24      47
AMD Athlon X2 4600+       2.4GHz  (VisualC6.0)          8      11      16      25      55
Intel Quad-Core Q6600    2.48Ghz (OpenWatcom1.8)       8      13      17      26      48
AMD Athlon X2 4600+       2.4GHz  (OpenWatcom1.8)       8      11      16      26      58
Intel Pentium M          1.8GHz  (VisualC6.0)          8      11      17      27      65
Intel Celeron M          1.0GHz  (VisualC10.0)         9       9      19      27      66
Intel Pentium M          1.8GHz  (OpenWatcom1.8)       8      10      18      29      67
Intel Celeron M          1.0GHz  (VisualC9.0)          9      11      20      27      66
Intel Core 2 Duo         1.5GHz  (VisualC6.0)          9      12      23      34      84
Intel Core 2 Duo         1.5GHz  (OpenWatcom1.8)       8      13      24      36      83
Intel Pentium 4          2.8GHz  (VisualC6.0)          8      12      22      40      90
Intel Celeron M          600MHz  (VisualC9.0)          8      14      24      44     108
Intel Pentium 4          2.8GHz  (OpenWatcom1.8)       9      13      24      44     101
Intel Celeron M          600MHz  (VisualC10.0)        10      14      25      44     108
Intel Celeron M          1.0GHz  (VisualC6.0)          8      14      27      46     153
Intel Celeron M          1.0GHz  (OpenWatcom1.8)      10      13      28      50     123
Intel Celeron M          600MHz  (VisualC6.0)          9      16      40      76     197
Intel Celeron M          600MHz  (OpenWatcom1.8)      10      17      46      89     209
Intel Pentium III        400MHz  (VisualC10.0)        12      17      50      96     259
Intel Pentium III        400MHz  (VisualC9.0)         13      16      51      97     264
Intel Pentium III        400MHz  (VisualC6.0)         12      22      70     146     378
Intel Pentium III        400MHz  (OpenWatcom1.8)      11      22      84     172     405
```

xpcbench this

Use the five predefined benchmarks to benchmark the target computer.

Start the target computer and run confidence test.

xpcbench_prework_snippet

Run five benchmark models and display results.

xpcbench this

```
### Starting xPC Target build procedure for model:
    xpcminimal
### Successful completion of build procedure for model:
```

```
        xpcminimal
### Looking for target: TargetPC1
### Download model onto target: TargetPC1


### Running benchmark for model: xpcminimal

### Starting xPC Target build procedure for model:
     f14tmp1
### Successful completion of build procedure for model:
     f14tmp1
### Looking for target: TargetPC1
### Download model onto target: TargetPC1


### Running benchmark for model: f14tmp1

### Starting xPC Target build procedure for model:
     f14tmp5
### Successful completion of build procedure for model:
     f14tmp5
### Looking for target: TargetPC1
### Download model onto target: TargetPC1


### Running benchmark for model: f14tmp5

### Starting xPC Target build procedure for model:
     f14tmp1O
### Successful completion of build procedure for model:
     f14tmp1O
### Looking for target: TargetPC1
### Download model onto target: TargetPC1


### Running benchmark for model: f14tmp1O

### Starting xPC Target build procedure for model:
     f14tmp25
### Successful completion of build procedure for model:
     f14tmp25
```

```
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: f14tmp25
```

xpcbench xpcosc

Use model xpcosc to benchmark the target computer.

Start the target computer and run confidence test.

xpcbench_prework_snippet

Run benchmark on xpcosc and print results.

xpcbench xpcosc

```
### Starting xPC Target build procedure for model:
    xpcosc
```

```
### Successful completion of build procedure for model:
    xpcosc
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: xpcosc

Benchmark results for model:              xpcosc
Number of blocks in model:            10
Elapsed time for model build (sec):       25.9
Elapsed time for model benchmark (sec):   23.4
Minimal achievable sample time (microsec): 11.9
```

xpcbench minimal

Use the minimal model to benchmark the target computer.

Start the target computer and run confidence test.

xpcbench_prework_snippet

Run benchmark on minimal model and print results.

xpcbench minimal

```
### Starting xPC Target build procedure for model:
    xpcminimal
### Generated code for 'xpcminimal' is up to date
### Successful completion of build procedure for model:
    xpcminimal
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: xpcminimal

Benchmark results for model:              Minimal
Number of blocks in model:            3
Elapsed time for model build (sec):       12.8
Elapsed time for model benchmark (sec):   23.4
```

```
Minimal achievable sample time (microsec): 8.5
```

xpcbench f14*5 -reboot

Benchmark the target computer with five copies of model f14, and then restart the computer.

Start the target computer and run confidence test.

```
xpcbench_prework_snippet
```

Run benchmark on f14*5, restart, and print results.

```
xpcbench f14*5 -reboot

### Starting xPC Target build procedure for model:
    f14tmp5
### Generated code for 'f14tmp5' is up to date
### Successful completion of build procedure for model:
    f14tmp5
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: f14tmp5
### Reboot target: TargetPC1...... OK

Benchmark results for model:              F14*5
Number of blocks in model:                310
Elapsed time for model build (sec):       14.6
Elapsed time for model benchmark (sec):   29.7
Minimal achievable sample time (microsec): 18.1
```

xpcbench f14*10 -reboot

Benchmark the target computer with ten copies of model f14, and then restart the computer.

Start the target computer and run confidence test.

```
xpcbench_prework_snippet
```

Run benchmark on `f14*10`, restart, and print results.

```
xpcbench f14*10 -reboot

### Starting xPC Target build procedure for model:
    f14tmp10
### Generated code for 'f14tmp10' is up to date
### Successful completion of build procedure for model:
    f14tmp10
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: f14tmp10
### Reboot target: TargetPC1...... OK

Benchmark results for model:              F14*10
Number of blocks in model:                620
Elapsed time for model build (sec):       18.3
Elapsed time for model benchmark (sec):   31.9
Minimal achievable sample time (microsec): 21.4
```

xpcbench this -cleanup

Benchmark the target computer with the five predefined benchmarks, and then delete build files.

Start the target computer and run confidence test.

```
xpcbench_prework_snippet
```

Run the five benchmark models, clean up build files, and display results.

```
xpcbench this -cleanup

### Starting xPC Target build procedure for model:
    xpcminimal
### Generated code for 'xpcminimal' is up to date
### Successful completion of build procedure for model:
    xpcminimal
```

```
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: xpcminimal

### Starting xPC Target build procedure for model:
    f14tmp1
### Generated code for 'f14tmp1' is up to date
### Successful completion of build procedure for model:
    f14tmp1
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: f14tmp1

### Starting xPC Target build procedure for model:
    f14tmp5
### Generated code for 'f14tmp5' is up to date
### Successful completion of build procedure for model:
    f14tmp5
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: f14tmp5

### Starting xPC Target build procedure for model:
    f14tmp10
### Generated code for 'f14tmp10' is up to date
### Successful completion of build procedure for model:
    f14tmp10
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: f14tmp10

### Starting xPC Target build procedure for model:
    f14tmp25
```

```
### Generated code for 'f14tmp25' is up to date
### Successful completion of build procedure for model:
    f14tmp25
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: f14tmp25
```

```
xPC Target Benchmark - Details                                          □ ⊡ ▣
File  Edit  View  Insert  Tools  Desktop  Window  Help                        ↘

                    Minimal achievable sample times in μ s

                                              Minimal   F14   F14*5  F14*10  F14*25
Intel Core 2 Duo E6420   2.13GHz (VisualC9.0)     8       9     11     14     21
Intel Core 2 Duo E6420   2.13GHz (VisualC10.0)    8      10     11     13     22
AMD Athlon X2 4600+      2.4GHz (VisualC9.0)       8       9     11     14     25
AMD Athlon X2 4600+      2.4GHz (VisualC10.0)      8       9     12     14     25
Intel Quad-Core Q6600    2.48Ghz (VisualC9.0)      8      12     13     16     30
Intel Quad-Core Q6600    2.48Ghz (VisualC10.0)     8      10     13     17     30
Intel Pentium M          1.8GHz (VisualC10.0)      8       8     13     18     36
Intel Pentium M          1.8GHz (VisualC9.0)       8      10     13     17     36
Intel Core 2 Duo E6420   2.13GHz (VisualC6.0)      8      10     13     19     33
Intel Core 2 Duo E6420   2.13GHz (OpenWatcom1.8)   8       9     14     19     33
Intel Pentium 4          2.8GHz (VisualC9.0)       8      10     14     17     37
Intel Pentium 4          2.8GHz (VisualC10.0)      8      10     13     18     38
Intel Core 2 Duo         1.5GHz (VisualC9.0)       9       9     18     23     43
This Machine                                      13      15     18     21     35
Intel Core 2 Duo         1.5GHz (VisualC10.0)      9      12     17     24     44
Intel Quad-Core Q6600    2.48Ghz (VisualC6.0)      8      13     18     24     47
AMD Athlon X2 4600+      2.4GHz (VisualC6.0)       8      11     16     25     55
Intel Quad-Core Q6600    2.48Ghz (OpenWatcom1.8)   8      13     17     26     48
AMD Athlon X2 4600+      2.4GHz (OpenWatcom1.8)    8      11     16     26     58
Intel Pentium M          1.8GHz (VisualC6.0)       8      11     17     27     65
Intel Celeron M          1.0GHz (VisualC10.0)      9       9     19     27     66
Intel Pentium M          1.8GHz (OpenWatcom1.8)    8      10     18     29     67
Intel Celeron M          1.0GHz (VisualC9.0)       9      11     20     27     66
Intel Core 2 Duo         1.5GHz (VisualC6.0)       9      12     23     34     84
Intel Core 2 Duo         1.5GHz (OpenWatcom1.8)    8      13     24     36     83
Intel Pentium 4          2.8GHz (VisualC6.0)       8      12     22     40     90
Intel Celeron M          600MHz (VisualC9.0)       8      14     24     44    108
Intel Pentium 4          2.8GHz (OpenWatcom1.8)    9      13     24     44    101
Intel Celeron M          600MHz (VisualC10.0)     10      14     25     44    108
Intel Celeron M          1.0GHz (VisualC6.0)       8      14     27     46    153
Intel Celeron M          1.0GHz (OpenWatcom1.8)   10      13     28     50    123
Intel Celeron M          600MHz (VisualC6.0)       9      16     40     76    197
Intel Celeron M          600MHz (OpenWatcom1.8)   10      17     46     89    209
Intel Pentium III        400MHz (VisualC10.0)     12      17     50     96    259
Intel Pentium III        400MHz (VisualC9.0)      13      16     51     97    264
Intel Pentium III        400MHz (VisualC6.0)      12      22     70    146    378
Intel Pentium III        400MHz (OpenWatcom1.8)   11      22     84    172    405
```

xpcbench xpcosc -cleanup

Benchmark the target computer with xpcosc, and then delete build files.

Start the target computer and run confidence test.

```
xpcbench_prework_snippet
```

Run the xpcosc, clean up build files, and display results.

```
xpcbench xpcosc -cleanup

### Starting xPC Target build procedure for model:
```

```
     xpcosc
### Generated code for 'xpcosc' is up to date
### Successful completion of build procedure for model:
     xpcosc
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: xpcosc

Benchmark results for model:            xpcosc
Number of blocks in model:              10
Elapsed time for model build (sec):     16.8
Elapsed time for model benchmark (sec): 23.4
Minimal achievable sample time (microsec): 9.9
```

xpcbench xpcosc -verbose

Use xpcosc to benchmark the target computer, printing build messages.

Start the target computer and run confidence test.

xpcbench_prework_snippet

Build the xpcosc model, print build messages, run the benchmark, and display results.

xpcbench xpcosc -verbose

```
### Starting xPC Target build procedure for model:
     xpcosc
### Generating code into build folder: xpcosc_xpc_rtw
### Invoking Target Language Compiler on xpcosc.rtw
### Using System Target File: xpctarget.tlc
### Loading TLC function libraries
.
.
.
### Created DLM ..\xpcosc.dlm
### Successful completion of build procedure for model:
```

```
     xpcosc
### Looking for target: TargetPC1
### Download model onto target: TargetPC1
### Create xPC Object tg

xPC Object

  Connected            = Yes
  Application          = xpcosc
  Mode                 = Real-Time Single-Tasking
  Status               = stopped
  CPUOverload          = none
.
.
.
### Running benchmark for model: xpcosc

Benchmark results for model:              xpcosc
Number of blocks in model:                10
Elapsed time for model build (sec):       28.6
Elapsed time for model benchmark (sec):   23.4
Minimal achievable sample time (microsec): 14.8
```

xpcbench f14*25 -cleanup -reboot

Benchmark the target computer with 25 copies of f14, restart the target computer, and then delete build files.

Start the target computer and run confidence test.

```
xpcbench_prework_snippet
```

Build model f14*25, run benchmark, restart, delete build files, and print results.

```
xpcbench f14*25 -cleanup -reboot

### Starting xPC Target build procedure for model:
    f14tmp25
```

```
### Successful completion of build procedure for model:
    f14tmp25
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: f14tmp25
### Reboot target: TargetPC1...... OK

Benchmark results for model:              F14*25
Number of blocks in model:                1550
Elapsed time for model build (sec):       48.8
Elapsed time for model benchmark (sec):   34.0
Minimal achievable sample time (microsec): 35.1
```

xpcbench f14 -verbose -cleanup -reboot

Benchmark the target computer with model f14, print build messages, delete build files, and then restart the target computer.

Start the target computer and run confidence test.

```
xpcbench_prework_snippet
```

Build model f14, print build messages, run benchmark, clean up build files, restart, and print results.

```
xpcbench f14 -verbose -cleanup -reboot

### Starting xPC Target build procedure for model:
    f14tmp1
### Generating code into build folder: f14tmp1_xpc_rtw
### Invoking Target Language Compiler on f14tmp1.rtw
### Using System Target File: xpctarget.tlc
### Loading TLC function libraries
.
.
.
### Created DLM ..\f14tmp1.dlm
### Successful completion of build procedure for model:
```

```
       f14tmp1
### Looking for target: TargetPC1
### Download model onto target: TargetPC1
### Create xPC Object tg

xPC Object

   Connected            = Yes
   Application          = f14tmp1
   Mode                 = Real-Time Single-Tasking
   Status               = stopped
   CPUOverload          = none
.
.
.
### Running benchmark for model: f14tmp1
### Reboot target: TargetPC1...... OK

Benchmark results for model:              F14*1
Number of blocks in model:            62
Elapsed time for model build (sec):       25.3
Elapsed time for model benchmark (sec):   23.4
Minimal achievable sample time (microsec): 15.8
```
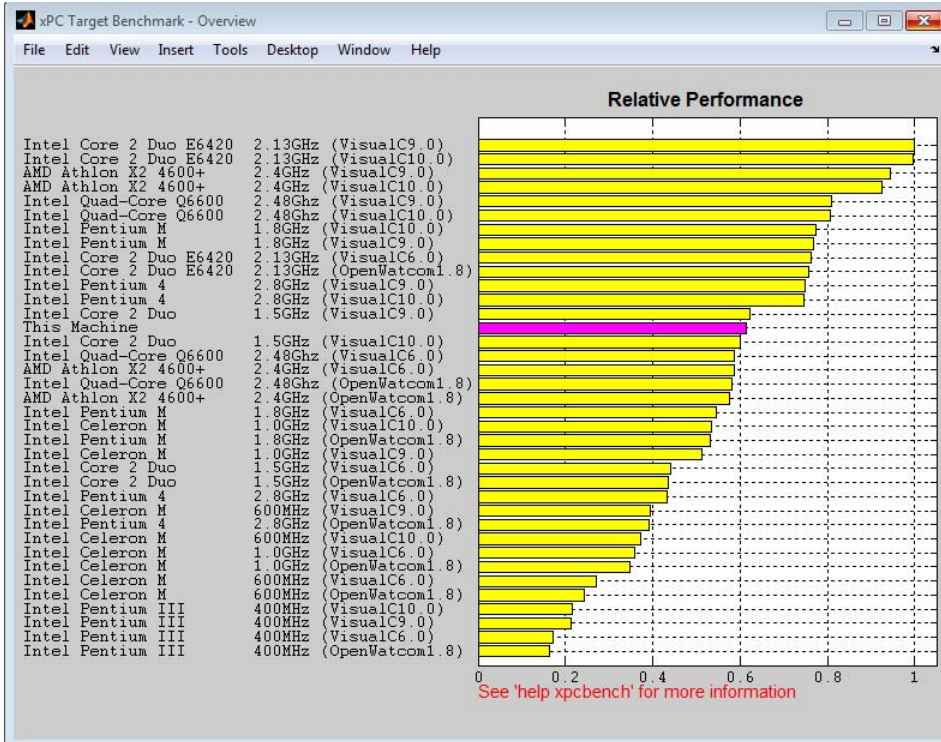
expected_results = xpcbench()

Return a structure array containing benchmark results showing what to expect of various target computers.

Start the target computer and run confidence test.

```
xpcbench_prework_snippet
```

Return an array with representative results for each processor type, in arbitrary order.

```
expected_results = xpcbench();
expected_results(1)
```

```
ans =

        Machine: 'Intel Celeron M
            600MHz (VisualC10.0)'
    BenchResults: [1.0000e-05 1.4000e-05 2.5000e-05 4.
            4000e-05 1.0800e-04]
            Desc: [1x70 char]
```

current_results = xpcbench('this','-cleanup');

Benchmark the target computer using the five predefined models, delete the build files, and return a structure array with results.

Start the target computer and run confidence test.

```
xpcbench_prework_snippet
```

Benchmark the target computer, delete the build files, and return a structure array.

```
current_results = xpcbench('this','-cleanup');
current_results(1)
```

```
### Starting xPC Target build procedure for model:
    xpcminimal
### Successful completion of build procedure for model:
    xpcminimal
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: xpcminimal

### Starting xPC Target build procedure for model:
    f14tmp1
### Successful completion of build procedure for model:
    f14tmp1
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: f14tmp1

### Starting xPC Target build procedure for model:
    f14tmp5
### Successful completion of build procedure for model:
    f14tmp5
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: f14tmp5

### Starting xPC Target build procedure for model:
    f14tmp1O
### Successful completion of build procedure for model:
    f14tmp1O
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: f14tmp1O

### Starting xPC Target build procedure for model:
```

```
       f14tmp25
### Successful completion of build procedure for model:
    f14tmp25
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: f14tmp25

ans =

        Name: 'Minimal'
      nBlocks: 3
    BuildTime: 23.8346
    BenchTime: 23.3861
        Tsmin: 1.3328e-05
```

xpcbench

Minimal achievable sample times in μ s

		Minimal	F14	F14*5	F14*10	F14*25
Intel Core 2 Duo E6420	2.13GHz (VisualC9.0)	8	9	11	14	21
Intel Core 2 Duo E6420	2.13GHz (VisualC10.0)	8	10	11	13	22
AMD Athlon X2 4600+	2.4GHz (VisualC9.0)	8	9	11	14	25
AMD Athlon X2 4600+	2.4GHz (VisualC10.0)	8	9	12	14	25
Intel Quad-Core Q6600	2.48GHz (VisualC9.0)	8	12	13	16	30
Intel Quad-Core Q6600	2.48Ghz (VisualC10.0)	8	10	13	17	30
Intel Pentium M	1.8GHz (VisualC10.0)	8	8	13	18	36
Intel Pentium M	1.8GHz (VisualC9.0)	8	10	13	17	36
Intel Core 2 Duo E6420	2.13GHz (VisualC6.0)	8	10	13	19	33
Intel Core 2 Duo E6420	2.13GHz (OpenWatcom1.8)	8	9	14	19	33
Intel Pentium 4	2.8GHz (VisualC9.0)	8	10	14	17	37
Intel Pentium 4	2.8GHz (VisualC10.0)	8	10	13	18	38
Intel Core 2 Duo	1.5GHz (VisualC9.0)	9	9	18	23	43
This Machine		13	15	18	21	35
Intel Core 2 Duo	1.5GHz (VisualC10.0)	9	12	17	24	44
Intel Quad-Core Q6600	2.48Ghz (VisualC6.0)	8	13	18	24	47
AMD Athlon X2 4600+	2.4GHz (VisualC6.0)	8	11	16	25	55
Intel Quad-Core Q6600	2.48Ghz (OpenWatcom1.8)	8	13	17	26	48
AMD Athlon X2 4600+	2.4GHz (OpenWatcom1.8)	8	11	16	26	58
Intel Pentium M	1.8GHz (VisualC6.0)	8	11	17	27	65
Intel Celeron M	1.0GHz (VisualC10.0)	9	9	19	27	66
Intel Pentium M	1.8GHz (OpenWatcom1.8)	8	10	18	29	67
Intel Celeron M	1.0GHz (VisualC9.0)	9	11	20	27	66
Intel Core 2 Duo	1.5GHz (VisualC6.0)	9	12	23	34	84
Intel Core 2 Duo	1.5GHz (OpenWatcom1.8)	8	13	24	36	83
Intel Pentium 4	2.8GHz (VisualC6.0)	8	12	22	40	90
Intel Celeron M	600MHz (VisualC9.0)	8	14	24	44	108
Intel Pentium 4	2.8GHz (OpenWatcom1.8)	9	13	24	44	101
Intel Celeron M	600MHz (VisualC10.0)	10	14	25	44	108
Intel Celeron M	1.0GHz (VisualC6.0)	8	14	27	46	153
Intel Celeron M	1.0GHz (OpenWatcom1.8)	10	13	28	50	123
Intel Celeron M	600MHz (VisualC6.0)	9	16	40	76	197
Intel Celeron M	600MHz (OpenWatcom1.8)	10	17	46	89	209
Intel Pentium III	400MHz (VisualC10.0)	12	17	50	96	259
Intel Pentium III	400MHz (VisualC9.0)	13	16	51	97	264
Intel Pentium III	400MHz (VisualC6.0)	12	22	70	146	378
Intel Pentium III	400MHz (OpenWatcom1.8)	11	22	84	172	405

current_results = xpcbench('f14','-verbose','-reboot','-cleanup')

Benchmark the target computer using the f14 model and all control options and return a structure array with results.

Start the target computer and run confidence test.

```
xpcbench_prework_snippet
```

Build 'f14', print build messages, run benchmark, delete build files, restart, and return results.

```
current_results = xpcbench('f14','-verbose','-reboot',
```

```
    '-cleanup')

### Starting xPC Target build procedure for model: f14tmp1
### Generating code into build folder: f14tmp1_xpc_rtw
### Invoking Target Language Compiler on f14tmp1.rtw
### Using System Target File: xpctarget.tlc
### Loading TLC function libraries
.
.
.
### Created DLM ..\f14tmp1.dlm
### Successful completion of build procedure for model:
     f14tmp1
### Looking for target: TargetPC1
### Download model onto target: TargetPC1
### Create xPC Object tg

xPC Object

  Connected           = Yes
  Application         = f14tmp1
  Mode                = Real-Time Single-Tasking
  Status              = stopped
  CPUOverload         = none
.
.
.
### Running benchmark for model: f14tmp1
### Reboot target: TargetPC1...... OK

Benchmark results for model:            F14*1
Number of blocks in model:              62
Elapsed time for model build (sec):     15.4
Elapsed time for model benchmark (sec): 23.4
Minimal achievable sample time (microsec): 15.8

current_results =
```

```
          Name: 'F14*1'
       nBlocks: 62
     BuildTime: 15.3662
     BenchTime: 23.3516
         Tsmin: 1.5750e-05
```

See Also   xpctest

External Web Sites   • http://www.mathworks.com/support/compilers/current_release/

xpcbootdisk

Purpose   Create xPC Target boot disk or DOS Loader files and confirm current environment properties

Syntax   xpcbootdisk

Description   xpcbootdisk creates an xPC Target boot floppy, CD or DVD boot image, network boot image, or DOS Loader files for the current xPC Target environment. Use the setxpcenv function to set environment properties.

- Creating an xPC Target boot floppy consists of writing the bootable kernel image onto the disk. You are asked to insert an empty formatted floppy disk into the drive. At the end, a summary of the creation process is displayed.

- Creating an xPC Target CD/DVD boot image consists of creating the bootable kernel image in a designated area. You can then burn the files to a blank CD/DVD. If you have Microsoft Windows Vista or Microsoft Windows XP Service Pack 2 or 3 with Image Mastering API v2.0 (IMAPIv2.0), xpcbootdisk offers to create to the CD or DVD. Otherwise, you must use alternate third-party CD/DVD writing software to write ISO® image files.

- Creating an xPC Target network boot image consists of running xpcnetboot to start the network boot server process.

- Creating xPC Target DOS Loader files consists of creating the files in a designated area. You can then copy the files to the target computer flash disk.

If you update the environment, you need to update the target boot floppy, CD boot image, network boot image, or DOS Loader files for the new xPC Target environment with the function xpcbootdisk.

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

xpcbootdisk

See Also   setxpcenv | getxpcenv

xpcbootdisk

How To

- "Target Boot Methods"
- "Command-Line Target Boot Methods" on page 4-39

Purpose   Generate file suitable for use by From File block

Syntax   xpcbytes2file(filename,var1,. . .,varn)

Arguments

filename	Name of the data file from which the From File block distributes data.
var1,. . .,varn	Column of data to be output to the model.

Description   xpcbytes2file(filename,var1,. . .,varn) outputs one column of var1, . . .,varn from file filename at every time step. All variables must have the same number of columns; the number of rows and data types can differ.

Note You might have the data organized such that a row refers to a single time step and not a column. In this case, pass to xpcbytes2file the transpose of the variable. To optimize file writes, organize the data in columns.

Examples   In the following example, to use the From File block to output a variable errorval (single precision, scalar) and velocity (double, width 3) at every time step, you can generate the file with the command:

xpcbytes2file('myfile', errorval, velocity)

where errorval has class 'single' and dimensions [1 x N] and velocity has class 'double' and dimensions [3 x N].

Set up the From File block to output

```
28 bytes
(1 * sizeof('single') + 3 * sizeof('double'))
```

at every sample time.

xpcexplr

Purpose   Open xPC Target Explorer

Syntax   xpcexplr

Description xpcexplr runs on the host computer and allows you to:

- Enter and change xPC Target environment properties
- Create an xPC Target bootable image
- Download, unload, and run target applications
- Monitor signals
- Tune parameters
- Add, remove, and configure xPC Target scopes
- Browse the target file system

See Also   setxpcenv | getxpcenv | xpcbootdisk

How To   • "Ethernet Communication Setup"
- "RS-232 Communication Setup"
- "Target Boot Methods"

Purpose   Compiler settings for xPC Target environment

Syntax

type = xpcgetCC
type = xpcgetCC('Type')
[*type*, *location*] = xpcgetCC
location= xpcgetCC('Location')
xpcgetCC('supported')
xpcgetCC('installed')
[*compilers*] = xpcgetCC('installed')

Description

type = xpcgetCC and *type* = xpcgetCC('Type') return the compiler type in *type*.

[*type*, *location*] = xpcgetCC returns the compiler type and its location in *type* and *location*.

location= xpcgetCC('Location') returns the compiler location in *location*.

xpcgetCC('supported') lists supported compiler versions for the xPC Target environment.

xpcgetCC('installed') lists the xPC Target supported compilers installed on the current host computer

[*compilers*] = xpcgetCC('installed') returns the xPC Target supported compilers installed on the current host computer in a structure.

Examples

Return the compiler type.

type = xpcgetCC

Return the compiler type and compiler location.

>> [type, location] = xpcgetCC

Return the xPC Target supported compilers installed on the current host computer in a structure and access the structure fields

```
[compilers] = xpcgetCC('installed')

compilers =

1x3 struct array with fields:
    Type
    Name
    Location

compilers.Type

ans =

VisualC
```

See Also   xpcsetCC

Purpose   Create kernel to boot target computer over dedicated network

Syntax   xpcnetboot
                  xpcnetboot *targetPCname*

Arguments | *targetPCName* | Target computer name as identified in xPC Target Explorer. |

Description   xpcnetboot creates an xPC Target kernel from which a target computer within the same network can start. This function also starts the following services as server processes:

- Bootstrap protocol (bootp) — xpcbootpserver.exe
- Trivial file transfer protocol (tftp) — xpctftpserver.exe

These processes respond to network boot requests from the target computer.

xpcnetboot without an argument creates a kernel for the default target computer (as identified in xPC Target Explorer).

xpcnetboot *targetPCname* creates an xPC Target kernel and waits for a request from the target computer named *targetPCname* (as identified in xPC Target Explorer).

Examples   In the following example, xpcnetboot creates an xPC Target kernel and waits for a request from the target computer, TargetPC1.

                  xpcnetboot TargetPC1

xpcsetCC

Purpose	Compiler settings for xPC Target environment

Syntax	`xpcsetCC('setup')`
	`xpcsetCC('`*location*`')`
	`xpcsetCC('`*type*`')`
	`xpcsetCC(`*type*`, `*location*`)`

Description	`xpcsetCC('setup')` queries the host computer for installed C compilers that the xPC Target environment supports. You can then select the C compiler. `xpcsetCC('`*location*`')` sets the compiler location. `xpcsetCC('`*type*`')` sets the compiler type. `'`*type*`'` must be `VISUALC`, representing the Microsoft Visual Studio C compiler. `xpcsetCC(`*type*`, `*location*`)` sets the compiler type and location.
See Also	`xpcgetCC`
How To	• "Command-Line C Compiler Configuration" on page 4-7

Purpose   Package for xPC Target MATLAB classes

Description   Use xpctarget package objects to access the MATLAB command line capabilities.

Functions

Assign these object creation functions to a MATLAB variable to get access to the properties and methods of the class.

Function	Description
xpctarget.fs	Create file system object
xpctarget.ftp	Create file transfer protocol (FTP) object
xpctarget.targets	Create container object to manage target computer environment collection objects
xpctarget.xpc	Create target object representing target application

xpctarget.env Class

Purpose   Stores target environment properties

Description   Each xpctarget.env Class object contains the environment properties for a particular target computer. A collection of these objects is stored in an xpctarget.targets Class object. An individual object in a collection is accessed via the xpctarget.targets.Item (env collection object) method.

Methods

Method	Description
xpctarget.env.get (env object)	Return property values for an environment object
xpctarget.env.set (env object)	Change property values for an environment object

Properties

The environment properties define communication between the host computer and target computer and the type of target boot floppy created during the setup process. An understanding of the environment properties will help you configure the xPC Target environment.

Tip To access a subset of these properties in xPC Target Explorer:

1 Expand a target computer node in the **Targets** pane.

2 Click the Target Properties icon in the toolbar or double-click **Properties**.

- Host-to-Target Communication on page 71
- Target Settings on page 77
- Boot Configuration on page 81

Host-to-Target Communication

Environment Property	Description
HostTargetComm	Property values are 'RS232' and 'TcpIp'.
	Select RS-232 or TCP/IP from the **Communication type** list in the **Target Properties** pane of xPC Target Explorer.
	If you select RS-232, you also must set the property RS232HostPort. If you select TCP/IP, then you also need to set all properties that start with TcpIp.

Environment Property	Description
	Note RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.
RS232Baudrate	Property values are '115200', '57600', '38400', '19200', '9600', '4800', '2400', and '1200'.
	Select 1200, 2400, 48 00, 9600, 19200, 38400, 57600, or 115200 from the **Baud rate** list in the **Target Properties** pane of xPC Target Explorer.
RS232HostPort	Property values are 'COM1' and 'COM2'.
	Select COM1 or COM2 from the **Host port** list in the **Target Properties** pane of xPC Target Explorer. The software automatically determines the COM port on the target computer.
	Before you can select an RS-232 port, you need to set the HostTargetComm property to RS232.

Environment Property	Description
`TcpIpGateway`	Property value is `'xxx.xxx.xxx.xxx'`. Enter the IP address for your gateway in the **Gateway** box in the **Target Properties** pane of xPC Target Explorer. This property is set by default to `255.255.255.255`, which means that a gateway is not used to connect to the target computer. If you communicate with your target computer from within a LAN that uses gateways, and your host and target computers are connected through a gateway, you must 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.
`TcpIpSubNetMask`	Property value is `'xxx.xxx.xxx.xxx'`. Enter the subnet mask of your LAN in the **Subnet mask** box in the **Target Properties** pane of xPC Target Explorer. Ask your system administrator for this value. For example, your subnet mask could be `255.255.255.0`.

Environment Property	Description
TcpIpTargetAddress	Property value is `'xxx.xxx.xxx.xxx'`. Enter a valid IP address for your target computer in the **IP address** box in the **Target Properties** pane of xPC Target Explorer. Ask your system administrator for this value. For example, `192.168.0.10`.
TcpIpTargetBusType	Property values are `'PCI'`, `'ISA'`, and `'USB'`. Select `PCI`, `ISA`, or `USB` from the **Bus type** list in the **Target Properties** pane of xPC Target Explorer. This property is set by default to `PCI`, and determines the bus type of your target computer. You do not need to define a bus type for your host computer, which can be the same or different from the bus type in your target computer. If `TcpIpTargetBusType` is set to `PCI`, then the properties `TcpIpISAMemPort` and `TcpIpISAIRQ` have no effect on TCP/IP communication. If you are using an ISA bus card, set `TcpIpTargetBusType` to `ISA` and enter values for `TcpIpISAMemPort` and `TcpIpISAIRQ`.

Environment Property	Description
TcpIpTargetDriver	Property values are '3C90x', 'I8254x', 'I82559', 'NE2000', 'NS83815', 'R8139', 'R8168', 'Rhine', 'RTLANCE', 'SMC91C9X', 'USBAX772', 'USBAX172', and 'Auto'.
	Select THREECOM_3C90x, INTEL_I8254x, INTEL_I82559, NE2000, NS83815, R8139, R8168, Rhine, RTLANCE, SMC91C9X, USBAX772, USBAX172, or Auto from the **Target driver** list in the **Target Properties** pane of xPC Target Explorer.
TcpIpTargetISAIRQ	Property value is 'n', where *n* is between 5 and 15 inclusive.
	Select an IRQ value from the **IRQ** list in the **Target Properties** pane of xPC Target Explorer.
	If you are using an ISA bus Ethernet card, you must enter values for the properties TcpIpISAMemPort and TcpIpISAIRQ. The values of these properties must correspond to the jumper settings or ROM settings on the ISA-bus Ethernet card.
	On your ISA bus card, assign an IRQ and I/O-port base address by moving the jumpers on the card.
	Set the IRQ to 5, 10, or 11. If one of these hardware settings

Environment Property	Description
	leads to a conflict in your target computer, choose another IRQ and make the corresponding changes to your jumper settings.
`TcpIpTargetISAMemPort`	Property value is `'0xnnnn'`.
	Enter an I/O port base address in the **Address** box in the **Target Properties** pane of xPC Target Explorer.
	If you are using an ISA bus Ethernet card, you must enter values for the properties `TcpIpISAMemPort` and `TcpIpISAIRQ`. The values of these properties must correspond to the jumper settings or ROM settings on your ISA bus Ethernet card.
	On your ISA bus card, assign an IRQ and I/O port base address by moving the jumpers on the card.
	Set the I/O port base address to around 0x300. If one of these hardware settings leads to a conflict in your target computer, choose another I/O port base address and make the corresponding changes to your jumper settings.
`TcpIpTargetPort`	Property value is `'xxxxx'`.
	Enter a port address greater than 20000 in the **Port** box in the

Environment Property	Description
	Target Properties pane of xPC Target Explorer.

This property is set by default to 22222. The default value is higher than the reserved area (telnet, ftp, . . .) and is only of use on the target computer. |

Target Settings

Environment Property	Description
EthernetIndex	Property value is 'n', where *n* indicates the index number for the Ethernet card on a target computer. Note that the (n-1)th Ethernet card on the target computer has an index number 'n'. The default index number is 0.

If the target computer has multiple Ethernet cards, you must select one of the cards for host-target communication. This option returns the index number of the card selected on the target computer upon booting. |
| LegacyMultiCoreConfig | Property values are 'on' and 'off' (the default).

Set this value to 'on' only if your target computer contains hardware not compliant with the Advanced Configuration and Power Interface (ACPI) standard. Otherwise, set this value to 'off'. |

Environment Property	Description
MaxModelSize	Property values are '1MB' (the default), '4MB', and '16MB'.
	Select 1 MB, 4 MB, or 16 MB from the **Model size** list in the **Target Properties** pane of xPC Target Explorer.
	Choosing the maximum model size reserves the specified amount of memory on the target computer for the target application. Memory not used by the target application is used by the kernel and by the heap for data logging.
	Selecting too high a value leaves less memory for data logging. Selecting too low a value does not reserve enough memory for the target application and creates an error. You can approximate the size of the target application by the size of the DLM file produced by the build process.
	• BootFloppy and DOSLoader modes ignore this value.
	• In StandAlone mode, you can only use MaxModelSize values '1MB' and '4MB'.
MulticoreSupport	Property values are 'on' and 'off' (the default).
	Select or clear the **Multicore CPU** check box in the **Target Properties** pane of xPC Target Explorer.
	If your target computer has multicore processors, set this value to 'on' to take advantage of these processors for background tasks. Otherwise, set this value to 'off'.
Name	Target computer name.

Environment Property	Description
NonPentiumSupport	Property values are 'on' and 'off' (the default).
	Select or clear the **Target is a 386/486** check box in the **Target Properties** pane of xPC Target Explorer.
	Set this value to 'on' if your target computer has a 386 or 486 compatible processor. Otherwise, set it to 'off'. If your target computer has a Pentium or higher compatible processor, selecting this check box slows the performance of your target computer.
SecondaryIDE	Property values are 'on' and 'off' (the default).
	Select or clear the **Secondary IDE** check box in the **Target Properties** pane of xPC Target Explorer.
	Set this value to 'on' only if you want to use the disks connected to a secondary IDE controller. If you do not have disks connected to the secondary IDE controller, leave this value set to 'off'.
ShowHardware	Property values are 'on' and 'off' (the default).
	If you create a target boot kernel when ShowHardware is 'on' and boot the target computer with it, the kernel displays the index, bus, slot, function, and target driver for each Ethernet card on the target monitor.
	The host computer cannot communicate with the target computer after the kernel boots with ShowHardware set.

Environment Property	Description
TargetRAMSizeMB	Property values are `'Auto'` (the default) and `'xxx'`, where *xxx* is a positive value specifying the amount of RAM, in megabytes, installed on the target computer.
	Under **RAM size**, click the **Auto** or **Manual** button in the **Target Properties** pane of xPC Target Explorer. If you click **Manual**, enter the amount of RAM, in megabytes, installed on the target computer in the **Size(MB)** box.
	`TargetRAMSizeMB` defines the total amount of installed RAM in the target computer. This RAM is used for the kernel, target application, data logging, and other functions that use the heap.
	If `TargetRAMSizeMB` is assigned `'Auto'`, the target application reads the target computer BIOS and determines the amount of memory up to a maximum of 2 GB. If the target application cannot read the BIOS, you must select Manual mode and enter the amount of memory, in megabytes, up to a maximum of 2 GB.
	The xPC Target kernel can use only 2 GB of memory.
TargetScope	Property values are `'Disabled'` and `'Enabled'` (the default).
	Select or clear the **Graphics mode** check box in the **Target Properties** pane of xPC Target Explorer.
	If you set `TargetScope` to `Disabled`, the target computer displays information as text.
	To use the full features of a target scope, install a keyboard on the target computer.
USBSupport	Property values are `'on'` (the default) and `'off'`.
	Select or clear the **USB Support** check box in the **Target Properties** pane of xPC Target Explorer.

Environment Property	Description
	Set this value to `'on'` if you want to use a USB port on the target computer; for example, to connect a USB mouse. Otherwise, set it to `'off'`.

Boot Configuration

Environment Property	Description
BootFloppyLocation	Drive name for creation of target boot disk.
CDBootImageLocation	Location of `cdboot.iso` file for creation of CD target boot disk.
DOSLoaderLocation	Location of DOSLoader files to boot target computers from devices other than floppy disk or CD.
EmbeddedOption	Property values are `'Disabled'` and `'Enabled'`. This property is read only. Note that the xPC Target Embedded Option product is enabled only if you purchase an additional license.

Environment Property	Description
TargetBoot	Property values are `'BootFloppy'`, `'CDBoot'`, `'DOSLoader'`, `'NetworkBoot'`, and `'StandAlone'`.
	Select Removable Disk, CD, DOS Loader, Network, or Stand Alone from the **Boot mode** list in the **Target Properties** pane of xPC Target Explorer.
	If your license file does not include the license for the xPC Target Embedded Option product, your only options are Removable Disk, CD, DOS Loader, and Network. With the xPC Target Embedded Option product licensed and installed, you have the additional choice of Stand Alone.
	Tip In the **Target Properties** pane of xPC Target Explorer, click the **Create boot disk** button to create a bootable image in the specified boot mode.
TargetMACAddress	Physical target computer MAC address from which to accept boot requests when booting within a dedicated network. Format the MAC address as six pairs of hexadecimal numbers, separated by colons:

Environment Property	Description
	xx:xx:xx:xx:xx:xx To update the MAC address in xPC Target Explorer, first click the **Reset** button in the **Target Properties** pane. You can then click the **Specify new MAC address** button to enter a MAC address manually in the **MAC address** box. If you do not enter a MAC address manually, the software will obtain the MAC address automatically the next time you restart the target computer.

xpctarget.env.get (env object)

Purpose   Return target environment property values

Syntax
```
property_value = env_object.property_name
property_value = env_object.get('property_name')
property_value = get(env_object,'property_name')
property_value = env_object.get
property_value = get(env_object)
```

Arguments

env_object	Name of a target environment object.
property_name	Name of a target environment object property.

Description   property_value = env_object.property_name gets the current value of property property_name from target environment object env_object. Alternative syntaxes are:

property_value = env_object.get('property_name')

property_value = get(env_object,'property_name')

property_value = env_object.get gets the values of all properties of target environment object env_object. An alternative syntax is:

property_value = get(env_object)

Get an individual environment object with the xpctarget.targets.Item (env collection object) method. For example:

```
tgs=xpctarget.targets;
env_object=tgs.Item('TargetPC1');
property_value=env_object.HostTargetComm
```

The environment properties for a target environment object are listed in the following tables.

Tip To access a subset of these properties in xPC Target Explorer:

1 Expand a target computer node in the **Targets** pane.

2 Click the Target Properties icon ![icon] in the toolbar or double-click **Properties**.

- "Host-to-Target Communication" on page 27-85
- "Target Settings" on page 27-91
- "Boot Configuration" on page 27-95

Host-to-Target Communication

Environment Property	Description
HostTargetComm	Property values are `'RS232'` and `'TcpIp'`.
	Select RS-232 or TCP/IP from the **Communication type** list in the **Target Properties** pane of xPC Target Explorer.
	If you select RS-232, you also must set the property RS232HostPort. If you select TCP/IP, then you also need to set all properties that start with TcpIp.

Environment Property	Description
	Note RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.
RS232Baudrate	Property values are '115200', '57600', '38400', '19200', '9600', '4800', '2400', and '1200'.
	Select 1200, 2400, 48 00, 9600, 19200, 38400, 57600, or 115200 from the **Baud rate** list in the **Target Properties** pane of xPC Target Explorer.
RS232HostPort	Property values are 'COM1' and 'COM2'.
	Select COM1 or COM2 from the **Host port** list in the **Target Properties** pane of xPC Target Explorer. The software automatically determines the COM port on the target computer.
	Before you can select an RS-232 port, you need to set the HostTargetComm property to RS232.

Environment Property	Description
TcpIpGateway	Property value is `'xxx.xxx.xxx.xxx'`. Enter the IP address for your gateway in the **Gateway** box in the **Target Properties** pane of xPC Target Explorer. This property is set by default to `255.255.255.255`, which means that a gateway is not used to connect to the target computer. If you communicate with your target computer from within a LAN that uses gateways, and your host and target computers are connected through a gateway, you must 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.
TcpIpSubNetMask	Property value is `'xxx.xxx.xxx.xxx'`. Enter the subnet mask of your LAN in the **Subnet mask** box in the **Target Properties** pane of xPC Target Explorer. Ask your system administrator for this value. For example, your subnet mask could be `255.255.255.0`.

Environment Property	Description
TcpIpTargetAddress	Property value is `'xxx.xxx.xxx.xxx'`.
	Enter a valid IP address for your target computer in the **IP address** box in the **Target Properties** pane of xPC Target Explorer. Ask your system administrator for this value.
	For example, `192.168.0.10`.
TcpIpTargetBusType	Property values are `'PCI'`, `'ISA'`, and `'USB'`.
	Select `PCI`, `ISA`, or `USB` from the **Bus type** list in the **Target Properties** pane of xPC Target Explorer. This property is set by default to `PCI`, and determines the bus type of your target computer. You do not need to define a bus type for your host computer, which can be the same or different from the bus type in your target computer.
	If `TcpIpTargetBusType` is set to `PCI`, then the properties `TcpIpISAMemPort` and `TcpIpISAIRQ` have no effect on TCP/IP communication.
	If you are using an ISA bus card, set `TcpIpTargetBusType` to `ISA` and enter values for `TcpIpISAMemPort` and `TcpIpISAIRQ`.

Environment Property	Description
TcpIpTargetDriver	Property values are '3C90x', 'I8254x', 'I82559', 'NE2000', 'NS83815', 'R8139', 'R8168', 'Rhine', 'RTLANCE', 'SMC91C9X', 'USBAX772', 'USBAX172', and 'Auto'. Select THREECOM_3C90x, INTEL_I8254x, INTEL_I82559, NE2000, NS83815, R8139, R8168, Rhine, RTLANCE, SMC91C9X, USBAX772, USBAX172, or Auto from the **Target driver** list in the **Target Properties** pane of xPC Target Explorer.
TcpIpTargetISAIRQ	Property value is 'n', where *n* is between 5 and 15 inclusive. Select an IRQ value from the **IRQ** list in the **Target Properties** pane of xPC Target Explorer. If you are using an ISA bus Ethernet card, you must enter values for the properties TcpIpISAMemPort and TcpIpISAIRQ. The values of these properties must correspond to the jumper settings or ROM settings on the ISA-bus Ethernet card. On your ISA bus card, assign an IRQ and I/O-port base address by moving the jumpers on the card. Set the IRQ to 5, 10, or 11. If one of these hardware settings

Environment Property	Description
	leads to a conflict in your target computer, choose another IRQ and make the corresponding changes to your jumper settings.
TcpIpTargetISAMemPort	Property value is `'0xnnnn'`.
	Enter an I/O port base address in the **Address** box in the **Target Properties** pane of xPC Target Explorer.
	If you are using an ISA bus Ethernet card, you must enter values for the properties `TcpIpISAMemPort` and `TcpIpISAIRQ`. The values of these properties must correspond to the jumper settings or ROM settings on your ISA bus Ethernet card.
	On your ISA bus card, assign an IRQ and I/O port base address by moving the jumpers on the card.
	Set the I/O port base address to around 0x300. If one of these hardware settings leads to a conflict in your target computer, choose another I/O port base address and make the corresponding changes to your jumper settings.
TcpIpTargetPort	Property value is `'xxxxx'`.
	Enter a port address greater than 20000 in the **Port** box in the

Environment Property	Description
	Target Properties pane of xPC Target Explorer.
	This property is set by default to `22222`. The default value is higher than the reserved area (`telnet`, `ftp`, . . .) and is only of use on the target computer.

Target Settings

Environment Property	Description
EthernetIndex	Property value is 'n', where *n* indicates the index number for the Ethernet card on a target computer. Note that the `(n-1)th` Ethernet card on the target computer has an index number `'n'`. The default index number is 0.
	If the target computer has multiple Ethernet cards, you must select one of the cards for host-target communication. This option returns the index number of the card selected on the target computer upon booting.
LegacyMultiCoreConfig	Property values are `'on'` and `'off'` (the default).
	Set this value to `'on'` only if your target computer contains hardware not compliant with the Advanced Configuration and Power Interface (ACPI) standard. Otherwise, set this value to `'off'`.

Environment Property	Description
MaxModelSize	Property values are `'1MB'` (the default), `'4MB'`, and `'16MB'`.
	Select 1 MB, 4 MB, or 16 MB from the **Model size** list in the **Target Properties** pane of xPC Target Explorer.
	Choosing the maximum model size reserves the specified amount of memory on the target computer for the target application. Memory not used by the target application is used by the kernel and by the heap for data logging.
	Selecting too high a value leaves less memory for data logging. Selecting too low a value does not reserve enough memory for the target application and creates an error. You can approximate the size of the target application by the size of the DLM file produced by the build process. • BootFloppy and DOSLoader modes ignore this value.
	• In StandAlone mode, you can only use `MaxModelSize` values `'1MB'` and `'4MB'`.
MulticoreSupport	Property values are `'on'` and `'off'` (the default).
	Select or clear the **Multicore CPU** check box in the **Target Properties** pane of xPC Target Explorer.
	If your target computer has multicore processors, set this value to `'on'` to take advantage of these processors for background tasks. Otherwise, set this value to `'off'`.
Name	Target computer name.

Environment Property	Description
NonPentiumSupport	Property values are 'on' and 'off' (the default).
	Select or clear the **Target is a 386/486** check box in the **Target Properties** pane of xPC Target Explorer.
	Set this value to 'on' if your target computer has a 386 or 486 compatible processor. Otherwise, set it to 'off'. If your target computer has a Pentium or higher compatible processor, selecting this check box slows the performance of your target computer.
SecondaryIDE	Property values are 'on' and 'off' (the default).
	Select or clear the **Secondary IDE** check box in the **Target Properties** pane of xPC Target Explorer.
	Set this value to 'on' only if you want to use the disks connected to a secondary IDE controller. If you do not have disks connected to the secondary IDE controller, leave this value set to 'off'.
ShowHardware	Property values are 'on' and 'off' (the default).
	If you create a target boot kernel when ShowHardware is 'on' and boot the target computer with it, the kernel displays the index, bus, slot, function, and target driver for each Ethernet card on the target monitor.
	The host computer cannot communicate with the target computer after the kernel boots with ShowHardware set.

Environment Property	Description
TargetRAMSizeMB	Property values are `'Auto'` (the default) and `'xxx'`, where *xxx* is a positive value specifying the amount of RAM, in megabytes, installed on the target computer.
	Under **RAM size**, click the **Auto** or **Manual** button in the **Target Properties** pane of xPC Target Explorer. If you click **Manual**, enter the amount of RAM, in megabytes, installed on the target computer in the **Size(MB)** box.
	TargetRAMSizeMB defines the total amount of installed RAM in the target computer. This RAM is used for the kernel, target application, data logging, and other functions that use the heap.
	If TargetRAMSizeMB is assigned `'Auto'`, the target application reads the target computer BIOS and determines the amount of memory up to a maximum of 2 GB. If the target application cannot read the BIOS, you must select Manual mode and enter the amount of memory, in megabytes, up to a maximum of 2 GB.
	The xPC Target kernel can use only 2 GB of memory.
TargetScope	Property values are `'Disabled'` and `'Enabled'` (the default).
	Select or clear the **Graphics mode** check box in the **Target Properties** pane of xPC Target Explorer.
	If you set TargetScope to Disabled, the target computer displays information as text.
	To use the full features of a target scope, install a keyboard on the target computer.
USBSupport	Property values are `'on'` (the default) and `'off'`.
	Select or clear the **USB Support** check box in the **Target Properties** pane of xPC Target Explorer.

Environment Property	Description
	Set this value to `'on'` if you want to use a USB port on the target computer; for example, to connect a USB mouse. Otherwise, set it to `'off'`.

Boot Configuration

Environment Property	Description
BootFloppyLocation	Drive name for creation of target boot disk.
CDBootImageLocation	Location of cdboot.iso file for creation of CD target boot disk.
DOSLoaderLocation	Location of DOSLoader files to boot target computers from devices other than floppy disk or CD.
EmbeddedOption	Property values are `'Disabled'` and `'Enabled'`. This property is read only. Note that the xPC Target Embedded Option product is enabled only if you purchase an additional license.

Environment Property	Description
TargetBoot	Property values are `'BootFloppy'`, `'CDBoot'`, `'DOSLoader'`, `'NetworkBoot'`, and `'StandAlone'`.
	Select Removable Disk, CD, DOS Loader, Network, or Stand Alone from the **Boot mode** list in the **Target Properties** pane of xPC Target Explorer.
	If your license file does not include the license for the xPC Target Embedded Option product, your only options are Removable Disk, CD, DOS Loader, and Network. With the xPC Target Embedded Option product licensed and installed, you have the additional choice of Stand Alone.
	Tip In the **Target Properties** pane of xPC Target Explorer, click the **Create boot disk** button to create a bootable image in the specified boot mode.
TargetMACAddress	Physical target computer MAC address from which to accept boot requests when booting within a dedicated network. Format the MAC address as six pairs of hexadecimal numbers, separated by colons:

Environment Property	Description
	xx:xx:xx:xx:xx:xx To update the MAC address in xC Target Explorer, first click the **Reset** button in the **Target Properties** pane. You can then click the **Specify new MAC address** button to enter a MAC address manually in the **MAC address** box. If you do not enter a MAC address manually, the software will obtain the MAC address automatically the next time you restart the target computer.

See Also   xpctarget.env.set (env object)

xpctarget.env.set (env object)

Purpose   Change target environment object property values

Syntax

```
env_object.property_name = property_value
env_object.set('prop_name1,'prop_value1','prop_name2',. . .)
set(env_object,'prop_name1','prop_value1','prop_name2',. . .)
```

Arguments

env_object	Name of a target environment object.
property_name	Name of a target environment object property.
property_value	Value for a target environment object property. Always use quotation marks for character strings; quotation marks are optional for numbers.

Description   `env_object.property_name = property_value` sets property `property_name` of target environment object `env_object` to `property_value`. Alternative syntaxes for one or more property-value pairs are:

```
env_object.set('prop_name1,'prop_value1','prop_name2',.  .
.)
```

```
set(env_object,'prop_name1','prop_value1','prop_name2',.  .
.)
```

Get an individual environment object with the xpctarget.targets.Item (env collection object) method. For example:

```
tgs=xpctarget.targets;
env_object=tgs.Item('TargetPC1');
env_object.HostTargetComm='RS232'
```

Not all properties are user writable. The writable properties for a target environment object are listed in the following table.

Tip To access a subset of these properties in xPC Target Explorer:

1 Expand a target computer node in the **Targets** pane.

2 Click the Target Properties icon in the toolbar or double-click **Properties**.

- "Host-to-Target Communication" on page 27-99
- "Target Settings" on page 27-105
- "Boot Configuration" on page 27-109

Host-to-Target Communication

Environment Property	Description
HostTargetComm	Property values are `'RS232'` and `'TcpIp'`.
	Select RS-232 or TCP/IP from the **Communication type** list in the **Target Properties** pane of xPC Target Explorer.
	If you select RS-232, you also must set the property RS232HostPort. If you select TCP/IP, then you also need to set all properties that start with TcpIp.

Environment Property	Description
	Note RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.
RS232Baudrate	Property values are '115200', '57600', '38400', '19200', '9600', '4800', '2400', and '1200'.
	Select 1200, 2400, 48 00, 9600, 19200, 38400, 57600, or 115200 from the **Baud rate** list in the **Target Properties** pane of xPC Target Explorer.
RS232HostPort	Property values are 'COM1' and 'COM2'.
	Select COM1 or COM2 from the **Host port** list in the **Target Properties** pane of xPC Target Explorer. The software automatically determines the COM port on the target computer.
	Before you can select an RS-232 port, you need to set the HostTargetComm property to RS232.

Environment Property	Description
TcpIpGateway	Property value is `'xxx.xxx.xxx.xxx'`.

Enter the IP address for your gateway in the **Gateway** box in the **Target Properties** pane of xPC Target Explorer. This property is set by default to `255.255.255.255`, which means that a gateway is not used to connect to the target computer.

If you communicate with your target computer from within a LAN that uses gateways, and your host and target computers are connected through a gateway, you must 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. |
| TcpIpSubNetMask | Property value is `'xxx.xxx.xxx.xxx'`.

Enter the subnet mask of your LAN in the **Subnet mask** box in the **Target Properties** pane of xPC Target Explorer. Ask your system administrator for this value.

For example, your subnet mask could be `255.255.255.0`. |

Environment Property	Description
TcpIpTargetAddress	Property value is `'xxx.xxx.xxx.xxx'`. Enter a valid IP address for your target computer in the **IP address** box in the **Target Properties** pane of xPC Target Explorer. Ask your system administrator for this value. For example, `192.168.0.10`.
TcpIpTargetBusType	Property values are `'PCI'`, `'ISA'`, and `'USB'`. Select `PCI`, `ISA`, or `USB` from the **Bus type** list in the **Target Properties** pane of xPC Target Explorer. This property is set by default to `PCI`, and determines the bus type of your target computer. You do not need to define a bus type for your host computer, which can be the same or different from the bus type in your target computer. If `TcpIpTargetBusType` is set to `PCI`, then the properties `TcpIpISAMemPort` and `TcpIpISAIRQ` have no effect on TCP/IP communication. If you are using an ISA bus card, set `TcpIpTargetBusType` to `ISA` and enter values for `TcpIpISAMemPort` and `TcpIpISAIRQ`.

Environment Property	Description
TcpIpTargetDriver	Property values are '3C90x', 'I8254x', 'I82559', 'NE2000', 'NS83815', 'R8139', 'R8168', 'Rhine', 'RTLANCE', 'SMC91C9X', 'USBAX772', 'USBAX172', and 'Auto'.
	Select THREECOM_3C90x, INTEL_I8254x, INTEL_I82559, NE2000, NS83815, R8139, R8168, Rhine, RTLANCE, SMC91C9X, USBAX772, USBAX172, or Auto from the **Target driver** list in the **Target Properties** pane of xPC Target Explorer.
TcpIpTargetISAIRQ	Property value is 'n', where *n* is between 5 and 15 inclusive.
	Select an IRQ value from the **IRQ** list in the **Target Properties** pane of xPC Target Explorer.
	If you are using an ISA bus Ethernet card, you must enter values for the properties TcpIpISAMemPort and TcpIpISAIRQ. The values of these properties must correspond to the jumper settings or ROM settings on the ISA-bus Ethernet card.
	On your ISA bus card, assign an IRQ and I/O-port base address by moving the jumpers on the card.
	Set the IRQ to 5, 10, or 11. If one of these hardware settings

Environment Property	Description
	leads to a conflict in your target computer, choose another IRQ and make the corresponding changes to your jumper settings.
`TcpIpTargetISAMemPort`	Property value is `'0xnnnn'`.
	Enter an I/O port base address in the **Address** box in the **Target Properties** pane of xPC Target Explorer.
	If you are using an ISA bus Ethernet card, you must enter values for the properties `TcpIpISAMemPort` and `TcpIpISAIRQ`. The values of these properties must correspond to the jumper settings or ROM settings on your ISA bus Ethernet card.
	On your ISA bus card, assign an IRQ and I/O port base address by moving the jumpers on the card.
	Set the I/O port base address to around 0x300. If one of these hardware settings leads to a conflict in your target computer, choose another I/O port base address and make the corresponding changes to your jumper settings.
`TcpIpTargetPort`	Property value is `'xxxxx'`.
	Enter a port address greater than 20000 in the **Port** box in the

Environment Property	Description
	Target Properties pane of xPC Target Explorer.
	This property is set by default to 22222. The default value is higher than the reserved area (telnet, ftp, . . .) and is only of use on the target computer.

Target Settings

Environment Property	Description
EthernetIndex	Property value is 'n', where *n* indicates the index number for the Ethernet card on a target computer. Note that the (n-1)th Ethernet card on the target computer has an index number 'n'. The default index number is 0.
	If the target computer has multiple Ethernet cards, you must select one of the cards for host-target communication. This option returns the index number of the card selected on the target computer upon booting.
LegacyMultiCoreConfig	Property values are 'on' and 'off' (the default).
	Set this value to 'on' only if your target computer contains hardware not compliant with the Advanced Configuration and Power Interface (ACPI) standard. Otherwise, set this value to 'off'.

Environment Property	Description
MaxModelSize	Property values are `'1MB'` (the default), `'4MB'`, and `'16MB'`.
	Select 1 MB, 4 MB, or 16 MB from the **Model size** list in the **Target Properties** pane of xPC Target Explorer.
	Choosing the maximum model size reserves the specified amount of memory on the target computer for the target application. Memory not used by the target application is used by the kernel and by the heap for data logging.
	Selecting too high a value leaves less memory for data logging. Selecting too low a value does not reserve enough memory for the target application and creates an error. You can approximate the size of the target application by the size of the DLM file produced by the build process. • BootFloppy and DOSLoader modes ignore this value.
	• In StandAlone mode, you can only use `MaxModelSize` values `'1MB'` and `'4MB'`.
MulticoreSupport	Property values are `'on'` and `'off'` (the default).
	Select or clear the **Multicore CPU** check box in the **Target Properties** pane of xPC Target Explorer.
	If your target computer has multicore processors, set this value to `'on'` to take advantage of these processors for background tasks. Otherwise, set this value to `'off'`.
Name	Target computer name.

Environment Property	Description
NonPentiumSupport	Property values are `'on'` and `'off'` (the default).
	Select or clear the **Target is a 386/486** check box in the **Target Properties** pane of xPC Target Explorer.
	Set this value to `'on'` if your target computer has a 386 or 486 compatible processor. Otherwise, set it to `'off'`. If your target computer has a Pentium or higher compatible processor, selecting this check box slows the performance of your target computer.
SecondaryIDE	Property values are `'on'` and `'off'` (the default).
	Select or clear the **Secondary IDE** check box in the **Target Properties** pane of xPC Target Explorer.
	Set this value to `'on'` only if you want to use the disks connected to a secondary IDE controller. If you do not have disks connected to the secondary IDE controller, leave this value set to `'off'`.
ShowHardware	Property values are `'on'` and `'off'` (the default).
	If you create a target boot kernel when ShowHardware is `'on'` and boot the target computer with it, the kernel displays the index, bus, slot, function, and target driver for each Ethernet card on the target monitor.
	The host computer cannot communicate with the target computer after the kernel boots with ShowHardware set.

Environment Property	Description
TargetRAMSizeMB	Property values are 'Auto' (the default) and 'xxx', where *xxx* is a positive value specifying the amount of RAM, in megabytes, installed on the target computer.
	Under **RAM size**, click the **Auto** or **Manual** button in the **Target Properties** pane of xPC Target Explorer. If you click **Manual**, enter the amount of RAM, in megabytes, installed on the target computer in the **Size(MB)** box.
	TargetRAMSizeMB defines the total amount of installed RAM in the target computer. This RAM is used for the kernel, target application, data logging, and other functions that use the heap.
	If TargetRAMSizeMB is assigned 'Auto', the target application reads the target computer BIOS and determines the amount of memory up to a maximum of 2 GB. If the target application cannot read the BIOS, you must select Manual mode and enter the amount of memory, in megabytes, up to a maximum of 2 GB.
	The xPC Target kernel can use only 2 GB of memory.
TargetScope	Property values are 'Disabled' and 'Enabled' (the default).
	Select or clear the **Graphics mode** check box in the **Target Properties** pane of xPC Target Explorer.
	If you set TargetScope to Disabled, the target computer displays information as text.
	To use the full features of a target scope, install a keyboard on the target computer.
USBSupport	Property values are 'on' (the default) and 'off'.
	Select or clear the **USB Support** check box in the **Target Properties** pane of xPC Target Explorer.

Environment Property	Description
	Set this value to `'on'` if you want to use a USB port on the target computer; for example, to connect a USB mouse. Otherwise, set it to `'off'`.

Boot Configuration

Environment Property	Description
BootFloppyLocation	Drive name for creation of target boot disk.
CDBootImageLocation	Location of cdboot.iso file for creation of CD target boot disk.
DOSLoaderLocation	Location of DOSLoader files to boot target computers from devices other than floppy disk or CD.
EmbeddedOption	Property values are `'Disabled'` and `'Enabled'`. This property is read only. Note that the xPC Target Embedded Option product is enabled only if you purchase an additional license.

Environment Property	Description
TargetBoot	Property values are `'BootFloppy'`, `'CDBoot'`, `'DOSLoader'`, `'NetworkBoot'`, and `'StandAlone'`.
	Select Removable Disk, CD, DOS Loader, Network, or Stand Alone from the **Boot mode** list in the **Target Properties** pane of xPC Target Explorer.
	If your license file does not include the license for the xPC Target Embedded Option product, your only options are Removable Disk, CD, DOS Loader, and Network. With the xPC Target Embedded Option product licensed and installed, you have the additional choice of Stand Alone.
	Tip In the **Target Properties** pane of xPC Target Explorer, click the **Create boot disk** button to create a bootable image in the specified boot mode.
TargetMACAddress	Physical target computer MAC address from which to accept boot requests when booting within a dedicated network. Format the MAC address as six pairs of hexadecimal numbers, separated by colons:

Environment Property	Description
	xx:xx:xx:xx:xx:xx To update the MAC address in xPC Target Explorer, first click the **Reset** button in the **Target Properties** pane. You can then click the **Specify new MAC address** button to enter a MAC address manually in the **MAC address** box. If you do not enter a MAC address manually, the software will obtain the MAC address automatically the next time you restart the target computer.

See Also   xpctarget.env.get (env object)

xpctarget.fs Class

Purpose   Manage the folders and files on the target computer

Description   This class includes the folder methods from xpctarget.fsbase Class
                   and implements file access methods used on the target computer.

Constructor

Constructor	Description
xpctarget.fs	Create file system object

Methods

These methods are inherited from xpctarget.fsbase Class.

Method	Description
xpctarget.fsbase.cd	Change folder on target computer
xpctarget.fsbase.dir	List contents of current folder on target computer
xpctarget.fsbase.mkdir	Make folder on target computer
xpctarget.fsbase.pwd	Current folder path of target computer
xpctarget.fsbase.rmdir	Remove folder from target computer

These methods are specific to class fs.

Method	Description
xpctarget.fs.diskinfo	Information about target computer drive
xpctarget.fs.fclose	Close open target computer file(s)
xpctarget.fs.fileinfo	Target computer file information
xpctarget.fs.filetable	Information about open files in target computer file system

Method	Description
xpctarget.fs.fopen	Open target computer file for reading
xpctarget.fs.fread	Read open target computer file
xpctarget.fs.fwrite	Write binary data to open target computer file
xpctarget.fs.getfilesize	Size of file on target computer
xpctarget.fs.removefile	Remove file from target computer

xpctarget.fs

Purpose	Create xPC Target file system object
Syntax	filesys_object = xpctarget.fs
	filesys_object = xpctarget.fs('mode','arg1','arg2')

Arguments

filesys_object	Variable name to reference the file system object.	
mode	Optionally, enter the communication mode:	

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

	TCPIP	Specify TCP/IP connection with target computer.
	RS232	Specify RS-232 connection with target computer.
arg1	Optionally, enter an argument based on the mode value:	
	IP address	If mode is 'TCPIP', enter the IP address of the target computer.
	COM port	If mode is 'RS232', enter the host COM port.
arg2	Optionally, enter an argument based on the mode value:	
	Port	If mode is 'TCPIP', enter the port number for the target computer.
	Baud rate	If mode is 'RS232', enter the baud rate for the connection between the host and target computer.

Description

Constructor of a file system object (xpctarget.fs Class). The file system object represents the file system on the target computer. You work with the file system by changing the file system object using methods.

filesys_object = xpctarget.fs constructs a file system object for the case where you have one target computer object or designate a target computer as the default one in your system.

filesys_object = xpctarget.fs('mode','arg1','arg2') constructs a file system object for the case where you have multiple target computers in your system or want to identify a target computer with the file system object.

Examples

In the following example, a file system object for a target computer with an RS-232 connection is created.

```
fs1=xpctarget.fs('RS232','COM1','115200')
```

Optionally, if you have an xpctarget.xpc object, you can construct an xpctarget.fs object by passing the xpctarget.xpc object variable to the xpctarget.fs constructor as an argument.

```
tg1=xpctarget.xpc('RS232','COM1','115200');
fs2=xpctarget.fs(tg1)
```

xpctarget.fs.diskinfo

Purpose   Information about target computer drive

Syntax   filesys_obj.diskinfo(target_PC_drive)
                diskinfo(filesys_obj,target_PC_drive)

Arguments

filesys_obj	Name of the xpctarget.fs file system object.
target_PC_drive	Name of the target computer drive for which to return information.

Description filesys_obj.diskinfo(target_PC_drive) returns disk information for the specified target computer drive. An alternative syntax is:

diskinfo(filesys_obj,target_PC_drive)

This is a method of xpctarget.fs objects called from the host computer.

Examples   Return disk information for the target computer C:\ drive for the file system object fsys.

```
diskinfo(fsys,'C:\') or fsys.diskinfo('C:\')
ans =
                  Label: 'SYSTEM '
            DriveLetter: 'C'
               Reserved: ''
           SerialNumber: 1.0294e+009
     FirstPhysicalSector: 63
                FATType: 32
               FATCount: 2
           MaxDirEntries: 0
          BytesPerSector: 512
        SectorsPerCluster: 4
           TotalClusters: 2040293
             BadClusters: 0
            FreeClusters: 1007937
                   Files: 19968
              FileChains: 22480
              FreeChains: 1300
         LargestFreeChain: 64349
```

xpctarget.fs.fclose

Purpose   Close open target computer file(s)

Syntax   fclose(filesys_obj,file_ID)
                filesys_obj.fclose(file_ID)

Arguments
filesys_obj	Name of the xpctarget.fs file system object.
file_ID	File identifier of the file to close.

Description Method of xpctarget.fs objects. From the host computer, closes
one or more open files in the target computer file system (except
standard input, output, and error). The file_ID argument is the file
identifier associated with an open file (see xpctarget.fs.fopen and
xpctarget.fs.filetable). You cannot have more than eight files open
in the file system.

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   fclose | xpctarget.fs.fopen | xpctarget.fs.fread |
xpctarget.fs.filetable | xpctarget.fs.fwrite

Purpose   Target computer file information

Syntax   ```
                   fileinfo(filesys_obj,file_ID)
                   filesys_obj.fileinfo(file_ID)
                   ```

**Arguments**      | | |
                   |---|---|
                   | filesys_obj | Name of the xpctarget.fs file system object. |
                   | file_ID | File identifier of the file for which to get file information. |

**Description**    Method of xpctarget.fs objects. From the host computer, gets the information for the file associated with file_ID.

**Examples**       Return file information for the file associated with the file identifier h in the file system object fsys.

```
fileinfo(fsys,h) or fsys.fileinfo(h)
ans =
                   FilePos: 0
            AllocatedSize: 12288
            ClusterChains: 1
       VolumeSerialNumber: 1.0450e+009
                 FullName: 'C:\DATA.DAT'
```

xpctarget.fs.filetable

Purpose	Information about open files in target computer file system
Syntax	`filetable(filesys_obj)` `filesys_obj.filetable`
Arguments	`filesys_obj` Name of the `xpctarget.fs` file system object.
Description	Method of `xpctarget.fs` objects. From the host computer, displays a table of the open files in the target computer file system. You cannot have more than eight files open in the file system.
Examples	Return a table of the open files in the target computer file system for the file system object `fsys`.

```
filetable(fsys) or fsys.filetable
ans =
Index   Handle  Flags     FilePos Name
------------------------------------------
    0  00060000  R__          8512 C:\DATA.DAT
    1  00080001  R__             0 C:\DATA1.DAT
    2  000A0002  R__          8512 C:\DATA2.DAT
    3  000C0003  R__          8512 C:\DATA3.DAT
    4  001E000S  R__             0 C:\DATA4.DAT
```

The table returns the open file handles in hexadecimal. To convert a handle to one that other `xpctarget.fs` methods, such as `fclose`, can use, use the `hex2dec` function.

```
h1 = hex2dec('001E0001'))
h1 =
1966081
```

To close that file, use the `xpctarget.fs` `fclose` method.

```
fsys.fclose(h1);
```

See Also   xpctarget.fs.fopen | xpctarget.fs.fclose

xpctarget.fs.fopen

Purpose   Open target computer file for reading

Syntax
```
file_ID = fopen(file_obj,'file_name')
file_ID = file_obj.fopen('file_name')
file_ID = fopen(file_obj,'file_name',permission)
file_ID = file_obj.fopen('file_name',permission)
```

Arguments

file_obj	Name of the xpctarget.fs object.
'file_name'	Name of the target computer to open.
permission	Values are 'r', 'w', 'a', 'r+', 'w+', or 'a+'. This argument is optional with 'r' as the default value.

Description   Method of xpctarget.fs objects. From the host computer, opens the specified filename on the target computer for binary access.

The permission argument values are

- 'r'

  Open the file for reading (default). The method does nothing if the file does not already exist.

- 'w'

  Open the file for writing. The method creates the file if it does not already exist.

- 'a'

  Open the file for appending to the file. Initially, the file pointer is at the end of the file. The method creates the file if it does not already exist.

- 'r+'

  Open the file for reading and writing. Initially, the file pointer is at the beginning of the file. The method does nothing if the file does not already exist.

- `'w+'`

  Open the file for reading and writing. The method empties the file first, if the file already exists and has content, and places the file pointer at the beginning of the file. The method creates the file if it does not already exist.

- `'a+'`

  Open the file for reading and appending to the file. Initially, the file pointer is at the beginning of the file. The method creates the file if it does not already exist.

You cannot have more than eight files open in the file system. This method returns the file identifier for the open file in `file_ID`. You use `file_ID` as the first argument to the other file I/O methods (such as `xpctarget.fs.fclose`, `xpctarget.fs.fread`, and `xpctarget.fs.fwrite`).

Examples   Open the file `data.dat` in the target computer file system object `fsys`. Assign the resulting file handle to a variable for reading.

```
h = fopen(fsys,'data.dat') or fsys.fopen('data.dat')
ans =
     2883584
d = fread(h);
```

See Also   `fopen` | `xpctarget.fs.fclose` | `xpctarget.fs.fread` | `xpctarget.fs.fwrite`

xpctarget.fs.fread

Purpose   Read open target computer file

Syntax   A = file_obj.fread(file_ID)
                A = fread(file_obj,file_ID)
                A = file_obj.fread(file_ID,offset,numbytes)
                A = fread(file_obj,file_ID,offset,numbytes)

Arguments

file_obj	Name of the xpctarget.fs object.
file_ID	File identifier of the file to read.
offset	Position from the beginning of the file from which fread can start to read.
numbytes	Maximum number of bytes fread can read.

Description   A = file_obj.fread(file_ID) reads binary data from the file
on the target computer and writes it into matrix A. The file_ID
argument is the file identifier associated with an open file (see
xpctarget.fs.fopen). An alternative syntax is:

A = fread(file_obj,file_ID)

A = file_obj.fread(file_ID,offset,numbytes) reads a block of
bytes from file_ID and writes the block into matrix A. An alternative
syntax is:

A = fread(file_obj,file_ID,offset,numbytes)

The offset argument specifies the position from the beginning of the
file from which this function can start to read. numbytes specifies the
maximum number of bytes to read.

To get a count of the total number of bytes read into A, use the following:

count = length(A);

length(A) might be less than the number of bytes requested if that number of bytes are not currently available. It is zero if the operation reaches the end of the file.

This is a method of xpctarget.fs objects called from the host computer.

Examples   Open the file data.dat in the target computer file system object fsys. Assign the resulting file handle to a variable for reading.

```
h=fsys.fopen('data.dat')
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   fread | xpctarget.fs.fclose | xpctarget.fs.fopen | xpctarget.fs.fwrite

xpctarget.fs.fwrite

Purpose	Write binary data to open target computer file
Syntax	fwrite(file_obj,file_ID,A) file_obj.fwrite(file_ID,A)

Arguments

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.

Description Method of xpctarget.fs objects. From the host computer, 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 xpctarget.fs.fopen). fwrite requires that the file be open with write permission.

Examples Open the file data.dat in the target computer file system object fsys. Assign the resulting file handle to a variable for writing.

```
h = fopen(fsys,'data.dat','w')
```

or

```
fsys.fopen('data.dat','w')

ans =
    2883584
d = fwrite(fsys,h,magic(5));
```

This writes the elements of matrix A to the file handle h. This content is written in column order.

See Also fwrite | xpctarget.fs.fclose | xpctarget.fs.fopen | xpctarget.fs.fread

Purpose   Size of file on target computer

Syntax   getfilesize(file_obj,file_ID)
                   file_obj.getfilesize(file_ID)

Arguments

file_obj	Name of the xpctarget.fs object.
file_ID	File identifier of the file to get the size of.

Description   Method of xpctarget.fs objects. From the host computer, gets the
                   size (in bytes) of the file identified by the file_ID file identifier on the
                   target computer file system. Use the xPC Target file object method
                   xpctarget.fs.fopen to open the file system object.

Examples   Get the size of the file identifier h for the file system object fsys.

                   getfilesize(fsys,h) or fsys.getfilesize(h)

See Also   xpctarget.fs.fopen

xpctarget.fs.readxpcfile

Purpose	Interpret raw data from xPC Target file format

Syntax

```
readxpcfile('filename')
file = readxpcfile(data)
```

Arguments

data_vector	Vector of uint8 bytes.
'filename'	File from which the vector of uint8 bytes is read. Vector is written

Description

readxpcfile('filename') converts the contents of 'filename' to double precision data representing the signals and timestamps.

file = readxpcfile(data) converts vector data_vector to double precision data representing the signals and timestamps.

readxpcfile converts xPC Target file format content (in bytes) to double precision data. It returns a structure that contains the following fields:

- version

  Not used.

- sector

  Not used.

- headersize

  Not used.

- numSignals

  Array of signal names.

- data

  Array of signal data.

- signalNames

Cell array of signal names.

A file scope creates the data. After you download the data from a target computer, use one of the following to read the data:

- To read the data after you download it to the target computer, use the fread function

- To download the data to the target computer and read it, use the xpctarget.fs.fread method.

Examples Use the xpctarget.fs object to convert data:

```
file=xpctarget.fs;
h=file.fopen('filename');
data=file.fread(h);
file.fclose(h);
file = readxpcfile(data);
```

Use the xpctarget.ftp object to copy the file from the target computer to the host computer, then read and convert the data.

```
xpcftp=xpctarget.ftp
xpcftp.get('filename')
datafile = readxpcfile('filename') % Convert the data
```

Use the xpctarget.ftp object to copy the file from the target computer to the host computer, then read and convert the data separately.

```
xpcftp=xpctarget.ftp
xpcftp.get('filename')
handle=fopen('filename')
data=fread(handle,'*uint8'); % Data should be read in uint8 format
fclose(handle);
data=data';
datafile = readxpcfile(data); % Convert the data
```

See Also xpctarget.ftp.get (ftp) | xpctarget.fs.fopen | xpctarget.fs.fread

xpctarget.fs.removefile

Purpose	Remove file from target computer

Syntax

```
removefile(file_obj,file_name)
file_obj.removefile(file_name)
```

Arguments

file_name	Name of the file to remove from the target computer file system.
file_obj	Name of the xpctarget.fs object.

Description　Method of xpctarget.fs objects. Removes a file from the target computer file system.

Note　You cannot recover this file once it is removed.

Examples　Remove the file data2.dat from the target computer file system fsys.

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

or

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

Purpose   Select target computer drive

Syntax   selectdrive(file_obj,'drive')
                  file_obj.selectdrive('drive')

Arguments
drive	Name of the drive to set.
file_obj	Name of the xpctarget.fs object.

Description   Method of xpctarget.fs objects. selectdrive sets the current drive of
                  the target computer to the specified string. Enter the drive string with
                  an extra backslash (\). For example, D:\\ for the D:\ drive.

Note Use the xpctarget.fsbase.cd method instead to get the same
behavior.

Examples   Set the current target computer drive to D:\.

                  selectdrive(fsys,'D:\\')

                  or

                  fsys.selectdrive('D:\\')

xpctarget.fsbase Class

Purpose   Base class of file system and file transfer protocol (FTP) classes

Description This class is the base class for `xpctarget.fs Class` and `xpctarget.ftp Class`. All methods are inherited by the derived classes. The constructor for this class is called implicitly when the constructors for the derived classes are called:

Methods

These methods are inherited by the derived classes.

Method	Description
`xpctarget.fsbase.cd`	Change folder on target computer
`xpctarget.fsbase.dir`	List contents of current folder on target computer
`xpctarget.fsbase.mkdir`	Make folder on target computer
`xpctarget.fsbase.pwd`	Current folder path of target computer
`xpctarget.fsbase.rmdir`	Remove folder from target computer

Purpose   Change folder on target computer

Syntax   cd(file_obj,target_PC_dir)
                 file_obj.cd(target_PC_dir)

Arguments

file_obj	Name of the xpctarget.ftp or xpctarget.fs object.
target_PC_dir	Name of the target computer folder to change to.

Description Method of xpctarget.fsbase, xpctarget.ftp, and xpctarget.fs objects. From the host computer, changes folder on the target computer.

Examples   Change folder from the current to one named logs for the file system object fsys.

cd(fsys,logs) or fsys.cd(logs)

Change folder from the current to one named logs for the FTP object f.

cd(f,logs) or f.cd(logs)

See Also   cd | xpctarget.fsbase.mkdir | xpctarget.fsbase.pwd

xpctarget.fsbase.dir

Purpose   List contents of current folder on target computer

Syntax   `dir(file_obj)`

Arguments

`file_obj`	Name of the `xpctarget.ftp` or `xpctarget.fs` object.

Description Method of `xpctarget.fsbase`, `xpctarget.ftp`, and `xpctarget.fs` objects. From the host computer, lists the contents of the current folder on the target computer.

To get the results in an M-by-1 structure, use a syntax like `ans=dir(file_obj)`. This syntax returns a structure like the following:

```
ans =
1x5 struct array with fields:
name
date
time
bytes
isdir
```

where

- `name` — Name of an object in the folder, shown as a cell array. The name, stored in the first element of the cell array, can have up to eight characters. The three-character file extension is stored in the second element of the cell array.

- `date` — Date of the last save of that object

- `time` — Time of the last save of that object

- `bytes` — Size in bytes of that object

- `isdir` — Logical value indicating that the object is (1) or is not (0) a folder

Examples   List the contents of the current folder for the file system object `fsys`.
You can also list the contents of the current folder for the FTP object `f`.

```
dir(fsys) or dir(f)
4/12/1998    20:00                222390        IO  SYS
 11/2/2003   13:54                     6      MSDOS SYS
 11/5/1998   20:01                 93880    COMMAND COM
 11/2/2003   13:54  <DIR>              0       TEMP
 11/2/2003   14:00                    33 AUTOEXEC BAT
  11/2/2003  14:00                   512 BOOTSECT DOS
  18/2/2003  16:33                  4512 SC1SIGNA DAT
  18/2/2003  16:17  <DIR>              0      FOUND  000
 29/3/2003   19:19                  8512      DATA DAT
 28/3/2003   16:41                  8512 DATADATA DAT
 28/3/2003   16:29                  4512 SC4INTEG DAT
  1/4/2003    9:28             201326592 PAGEFILE SYS
 11/2/2003   14:13  <DIR>              0      WINNT
    4/5/2001  13:05                214432 NTLDR         '
  4/5/2001   13:05                 34468 NTDETECT COM
 11/2/2003   14:15  <DIR>              0   DRIVERS
  22/1/2001  11:42                   217      BOOT   INI'
 28/3/2003   16:41                  8512         A DAT
 29/3/2003   19:19                  2512 SC3SIGNA DAT
 11/2/2003   14:25  <DIR>              0   INETPUB
 11/2/2003   14:28                     0    CONFIG SYS
 29/3/2003   19:10                  2512 SC3INTEG DAT
  1/4/2003   18:05                  2512   SC1GAIN DAT
   11/2/2003  17:26  <DIR>              0 UTILIT~1
```

You must use the `dir(f)` syntax to list the contents of the folder.

See Also   `dir | xpctarget.fsbase.mkdir | xpctarget.fsbase.cd |`
`xpctarget.fsbase.pwd`

xpctarget.fsbase.mkdir

Purpose	Make folder on target computer

Syntax

```
mkdir(file_obj,dir_name)
file_obj.mkdir(dir_name)
```

Arguments

file_obj	Name of the xpctarget.ftp or xpctarget.fs object.
dir_name	Name of the folder to be created.

Description   Method of xpctarget.fsbase, xpctarget.ftp, and xpctarget.fs objects. From the host computer, makes a new folder in the current folder on the target computer file system.

Note that to delete a folder from the target computer, you need to reboot the computer into DOS or some other operating system and use a utility in that system to delete the folder.

Examples   Create a new folder, logs, in the target computer file system object fsys.

```
mkdir(fsys,logs)
```

or

```
fsys.mkdir(logs)
```

Create a new folder, logs, in the target computer FTP object f.

mkdir(f,logs) or f.mkdir(logs)

See Also   mkdir | xpctarget.fsbase.dir | xpctarget.fsbase.pwd

Purpose   Current folder path of target computer

Syntax   pwd(file_obj)
                  file_obj.pwd

Arguments   file_obj   Name of the xpctarget.ftp or xpctarget.fs object.

Description   Method of xpctarget.fsbase, xpctarget.ftp, and xpctarget.fs
                  objects. Returns the pathname of the current target computer folder.

Examples   Return the target computer current folder for the file system object
                  fsys.

                  pwd(fsys) or fsys.pwd

                  Return the target computer current folder for the FTP object f.

                  pwd(f) or f.pwd

See Also   pwd | xpctarget.fsbase.dir | xpctarget.fsbase.mkdir

xpctarget.fsbase.rmdir

Purpose   Remove folder from target computer

Syntax   rmdir(file_obj,dir_name)
                  file_obj.rmdir(dir_name)

Arguments

dir_name	Name of the folder to remove from the target computer file system.
file_obj	Name of the xpctarget.fs object.

Description   Method of xpctarget.fsbase, xpctarget.ftp, and xpctarget.fs objects. Removes a folder from the target computer file system.

Note You cannot recover this folder once it is removed.

Examples   Remove the folder data2dir.dat from the target computer file system fsys.

                  rmdir(f,'data2dir.dat')

                  or

                  fsys.rmdir('data2dir.dat')

Purpose   Manage the folders and files on the target computer via file transfer protocol (FTP)

Description The FTP object represents the file on the target computer. You work with the file folders using the inherited methods, and transport the file between the host and target computers using the xpctarget.ftp methods.

Constructor

Constructor	Description
xpctarget.ftp	Create file transfer protocol (FTP) object

Methods

These methods are inherited from xpctarget.fsbase Class.

Method	Description
xpctarget.fsbase.cd	Change folder on target computer
xpctarget.fsbase.dir	List contents of current folder on target computer
xpctarget.fsbase.mkdir	Make folder on target computer
xpctarget.fsbase.pwd	Current folder path of target computer
xpctarget.fsbase.rmdir	Remove folder from target computer

These methods are specific to class ftp.

Method	Description
xpctarget.ftp.get (ftp)	Retrieve copy of requested file from target computer
xpctarget.ftp.put	Copy file from host computer to target computer

xpctarget.ftp

Purpose	Create file transfer protocol (FTP) object
Syntax	file_object = xpctarget.ftp('mode', 'arg1', 'arg2')

Arguments

file_object	Variable name to reference the FTP object.	
mode	Optionally, enter the communication mode:	

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

	TCPIP	Specify TCP/IP connection with target computer.
	RS232	Specify RS-232 connection with target computer.
arg1	Optionally, enter an argument based on the mode value:	
	IP address	If mode is 'TCPIP', enter the IP address of the target computer.
	COM port	If mode is 'RS232', enter the host COM port.
arg2	Optionally, enter an argument based on the mode value:	
	Port	If mode is 'TCPIP', enter the port number for the target computer.
	Baud rate	If mode is 'RS232', enter the baud rate for the connection between the host and target computer.

Description Constructor of an FTP object (xpctarget.ftp Class). The FTP object represents the file on the target computer. You work with the file by changing the file object using methods.

If you have one target computer object, or if you designate a target computer as the default one in your system, use the syntax

```
file_object=xpctarget.ftp
```

If you have multiple target computers in your system, or if you want to identify a target computer with the file object, use the following syntax to create the additional file objects.

```
file_object=xpctarget.ftp('mode', 'arg1', 'arg2')
```

Examples   In the following example, a file object for a target computer with an RS-232 connection is created.

```
f=xpctarget.ftp('RS232','COM1','115200')

f =
 xpctarget.ftp
```

Optionally, if you have an xpctarget.xpc object, you can construct an xpctarget.ftp object by passing the xpctarget.xpc object variable to the xpctarget.ftp constructor as an argument.

```
>> tg1=xpctarget.xpc('RS232','COM1','115200');
>> f2=xpctarget.ftp(tg1)

f2 =

 xpctarget.ftp
```

xpctarget.ftp.get (ftp)

Purpose   Retrieve copy of requested file from target computer

Syntax   get(file_obj,file_name)
                 file_obj.get(file_name)

Arguments

file_obj	Name of the xpctarget.ftp object.
file_name	Name of a file on the target computer.

Description Method of xpctarget.ftp objects. Copies the specified filename
                 from the target computer to the current folder of the host computer.
                 file_name must be either a fully qualified filename on the target
                 computer, or located in the current folder of the target computer.

Examples   Retrieve a copy of the file named data.dat from the current folder of
                 the target computer file object f.

                 get(f,'data.dat') or f.get('data.dat')
                 ans = data.dat

See Also   xpctarget.ftp.put

Purpose   Copy file from host computer to target computer

Syntax   put(file_obj,file_name)
                   file_obj.put(file_name)

Arguments
file_obj	Name of the xpctarget.ftp object.
file_name	Name of the file to copy to the target computer.

Description   Method of xpctarget.ftp objects. Copies a file from the host computer to the target computer. file_name must be a file in the current folder of the host computer. The method writes file_name to the target computer disk.

                   put might be slower than the get operation for the same file. This is expected behavior.

Examples   Copy the file data2.dat from the current folder of the host computer to the current folder of the target computer FTP object f.

                   put(f,'data2.dat')

                   or

                   fsys.put('data2.dat')

See Also   xpctarget.fsbase.dir | xpctarget.ftp.get (ftp)

xpctarget.targets Class

Purpose   Container object to manage target computer environment collection objects

Description   The targets class contains a collection of environment settings, stored in xpctarget.env Class objects.

Constructor

Constructor	Description
xpctarget.targets	Create container object to manage target computer environment collection objects

Methods

Method	Description
xpctarget.targets.Add (env collection object)	Add a new xPC Target environment collection object.
xpctarget.targets.getTargetNames (env collection object)	Retrieve the xPC Target environment collection object names.
xpctarget.targets.Item (env collection object)	Retrieve xPC Target environment collection object.
xpctarget.targets.makeDefault (env collection object)	Set target computer environment collection object as default.
xpctarget.targets.Remove (env collection object)	Remove an xPC Target environment collection object.

Properties

Property	Description	Writable
DefaultTarget	Returns an xpctarget.env object that references the default target computer object environment.	No
NumTargets	Returns the number of target computer environment objects in the container.	No

xpctarget.targets

Purpose	Create container object to manage target computer environment collection objects
Syntax	env_collection_object = xpctarget.targets
Description	Constructor for target environment object collection (xpctarget.targets Class). The collection manages the environment object (xpctarget.env Class) for a multitarget xPC Target system. (This is in contrast to the setxpcenv and getxpcenv functions, which manage the environment properties for the default target computer.) You work with the environment objects by changing the environment properties using methods.

Use the syntax

env_object = xpctarget.targets

Access properties of an env_collection_object object with env_collection_object.propertyname, env_collection_object.propertyname.propertyname, or with the xpctarget.targets.get (env collection object) and xpctarget.targets.set (env collection object) commands.

Access an individual environment object via xpctarget.targets.Item (env collection object),

Examples	Create an environment container object. With this object, you can manage the environment collection objects for the targets in your system.

tgs=xpctarget.targets

See Also	xpctarget.targets.get (env collection object) | xpctarget.targets.set (env collection object)

xpctarget.targets.Add (env collection object)

Purpose   Add new xPC Target environment collection object

Syntax   `env_collection_object.Add`

Description Method of `xpctarget.targets` objects. `Add` creates an xPC Target environment collection object on the host computer.

Examples   Add a new xPC Target environment collection object to the system. Assume that `tgs` represents the environment collection object. The first `get(tgs)` function returns the current number of target computers. The second function returns the number of target computers after you add one.

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

See Also   `xpctarget.targets` | `xpctarget.targets.set (env collection object)` | `xpctarget.targets.get (env collection object)`

xpctarget.targets.get (env collection object)

Purpose   Return target object collection environment property values

Syntax   `get(env_collection_object, 'env_collection_object_property')`

Arguments

env_collection_object	Name of a collection of target objects.
'env_collection_object_property'	Name of a target object environment property.

Description   `get` gets the values of environment properties for a collection of target objects.

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

Property	Description	Writable
DefaultTarget	Contains an instance of the default target environment object (xpctarget.env).	No
NumTargets	Contains the number of target objects in the xPC Target system. Note that this is not the actual number of target computers in the system.	No

Examples   List the values of the target object collection environment property values. Assume that `tgs` represents the target object collection environment.

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

List the value for the target object environment collection property `NumTargets`. Note that the property name is a string, in quotation marks, and not case sensitive.

```
get(tgs,'NumTargets') or tgs.get('NumTargets'))
```

See Also   get | xpctarget.targets.set (env collection object) | set

xpctarget.targets.getTargetNames (env collection object)

Purpose	Retrieve xPC Target environment object names
Syntax	env_collection_object.getTargetNames
Description	Method of xpctarget.targets objects. getTargetNames retrieves the names of the existing xPC Target environment collection objects from the xpctarget.targets class.
Examples	Retrieve the names of the xPC Target environment collection objects in the system. Assume that tgs represents the target object collection environment.

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

See Also	xpctarget.targets	xpctarget.targets.set (env collection object)	xpctarget.targets.get (env collection object)

xpctarget.targets.Item (env collection object)

Purpose   Retrieve specific xPC Target environment (env) object

Syntax   env_collection_object.Item('env_object_name')

Description   Method of xpctarget.targets objects. Item retrieves a
specific environment object (xpctarget.env Class) from the
xpctarget.targets class. Use this method to work with a particular
target computer environment object.

Examples   Retrieve a new xPC Target environment collection object from the
system. Assume that tgs represents the target object collection
environment.

```
tgs=xpctarget.targets;
get(tgs);
tgs.getTargetNames
tgs.Item('TargetPC1')
```

See Also   xpctarget.targets | xpctarget.targets.set (env collection
object) | xpctarget.targets.get (env collection object)

xpctarget.targets.makeDefault (env collection object)

Purpose	Set specific target computer environment object as default
Syntax	env_collection_object.makeDefault(`env_object_name')
Description	Method of xpctarget.targets objects. makeDefault sets the specified target computer environment object as the default target computer from the xpctarget.targets class.
Examples	Set the specified target collection object as the default target computer collection. Assume that tgs represents the target object collection environment.

```
tgs=xpctarget.targets;
get(tgs);
tgs.getTargetNames
tgs.makeDefault('TargetPC2')
```

See Also   xpctarget.targets | xpctarget.targets.set (env collection object) | xpctarget.targets.get (env collection object)

xpctarget.targets.Remove (env collection object)

Purpose	Remove specific xPC Target environment object
Syntax	env_collection_object.Remove('env_collection_object_name')
Description	Method of xpctarget.targets objects. Remove removes an existing xPC Target environment object from the environment collection. If you remove the target environment object of the default target computer, the next target environment object becomes the default target computer. You can remove all but the last target computer, which becomes the default target computer.
Examples	Remove an xPC Target environment collection object from the system. Assume that tgs represents the target object collection environment.

```
tgs=xpctarget.targets;
get(tgs);
tgs.getTargetNames
tgs.Remove('TargetPC2')
```

See Also   xpctarget.targets | xpctarget.targets.set (env collection object) | xpctarget.targets.get (env collection object)

xpctarget.targets.set (env collection object)

Purpose   Change target object environment collection object property values

Syntax
```
set(env_collection_object)
set(env_collection_object, 'property_name1',
'property_value1','property_name2', 'property_value2', . . .)
env_collection_object.set('property_name1',
'property_value1')
set(env_collection_object, property_name_vector,
property_value_vector)
env_collection_object.property_name = property_value
```

Arguments

env_collection_object	Name of a target environment collection object.
'property_name'	Name of a target object environment collection property. Always use quotation marks for character strings.
property_value	Value for a target object environment collection property. Always use quotation marks for character strings; quotation marks are optional for numbers.

Description   set sets the values of environment properties for a collection of target object environments. Not all properties are user writable. Properties are entered as property-value pairs.

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

Property	Description	Writable
DefaultTarget	Contains an instance of the default target environment object (xpctarget.env).	No
NumTargets	Contains the number of target objects in the xPC Target system. Note that this is not the actual number of target computers in the system.	No

See Also   get | set | xpctarget.targets.get (env collection object)

xpctarget.xpc Class

Purpose   Target object representing target application

Description Provides access to methods and properties used to start and stop the target application, read and set parameters, monitor signals, and retrieve status information about the target computer.

Constructor

Constructor	Description
xpctarget.xpc	Create target object representing target application

Methods

Method	Description
xpctarget.xpc.addscope	Create scopes
xpctarget.xpc.close	Close serial port connecting host computer with target computer
xpctarget.xpc.get (target application object)	Return target application object property values
xpctarget.xpc.getlog	All or part of output logs from target object
xpctarget.xpc.getparam	Value of target object parameter index
xpctarget.xpc.getparamid	Parameter index from parameter list
xpctarget.xpc.getparamname	Block path and parameter name from index list
xpctarget.xpc.getscope	Scope object pointing to scope defined in kernel
xpctarget.xpc.getsignal	Value of target object signal index
xpctarget.xpc.getsignalid	Signal index or signal property from signal list
xpctarget.xpc.getsignalidsfromlabel	Return vector of signal indices
xpctarget.xpc.getsignallabel	Return signal label
xpctarget.xpc.getsignalname	Signal name from index list

Method	Description
xpctarget.xpc.getxpcpci	Determine which PCI boards are installed in target computer
xpctarget.xpc.load	Download target application to target computer
xpctarget.xpc.loadparamset	Restore parameter values saved in specified file
xpctarget.xpc.reboot	Reboot target computer
xpctarget.xpc.remscope	Remove scope from target computer
xpctarget.xpc.saveparamset	Save current target application parameter values
xpctarget.xpc.set (target application object)	Change target application object property values
xpctarget.xpc.setparam	Change writable target object parameters
xpctarget.xpc.start (target application object)	Start execution of target application on target computer
xpctarget.xpc.stop (target application object)	Stop execution of target application on target computer
xpctarget.xpc.targetping	Test communication between host and target computers
xpctarget.xpc.unload	Remove current target application from target computer

Properties

Properties are read using xpctarget.xpc.get (target application object). Writable properties are written using xpctarget.xpc.set (target application object).

Property	Description	Writable
Application	Name of the Simulink model and target application built from that model.	No
AvgTET	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.	No
	The TET includes:	
	• Complete I/O latency.	
	• Data logging (the parts that happen in a real-time task). This includes data captured in scopes.	
	• Asynchronous interruptions.	
	• Parameter updating latency (if the **Double buffer parameter changes** parameter is set in the **xPC Target options** node of the model Configuration Parameters dialog box).	
	Note that the TET is not the only consideration in determining the minimum achievable sample time. Other considerations, not included in the TET, are:	
	• Time required to measure TET	
	• Interrupt latency required to schedule and run one step of the model	

Property	Description	Writable
CommunicationTimeOut	Communication timeout between host and target computer, in seconds.	Yes
Connected	Communication status between the host computer and the target computer. Values are 'Yes' and 'No'.	No
CPUoverload	CPU status for overload. If the target application requires more CPU time than the sample time of the model, this value is set from 'none' to 'detected' and the current run is stopped. Returning this status to 'none' requires either a faster processor or a larger sample time.	No
ExecTime	Execution time. Time, in seconds, since your target application started running. When the target application stops, the total execution time is displayed.	No
LogMode	Controls which data points are logged: • Time-equidistant logging. Logs a data point at every time interval. Set value to 'Normal'. • 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.	Yes

Property	Description	Writable
MaxLogSamples	Maximum number of samples for each logged signal within the circular buffers for TimeLog, StateLog, OutputLog, and TETLog. StateLog and OutputLog can have one or more signals. This value is calculated by dividing the **Signal Logging Buffer Size** by the number of logged signals. The **Signal Logging Buffer Size** box is in the **xPC Target options** pane of the Configuration Parameters dialog box.	No
MaxTET	Maximum task execution time. Corresponds to the slowest time (longest time measured), in seconds, to update model equations and post outputs.	No
MinTET	Minimum task execution time. Corresponds to the fastest time (smallest time measured), in seconds, to update model equations and post outputs.	No
Mode	Type of Simulink Coder code generation. Values are 'Real-Time Singletasking', 'Real-Time Multitasking', and 'Accelerate'. The default value is 'Real-Time Singletasking'. Even if you select 'Real-Time Multitasking', the actual mode can be 'Real-Time Singletasking'. This happens if your model contains only one or two tasks and the sample rates are equal.	No

Property	Description	Writable
NumLogWraps	The number of times the circular buffer wrapped. The buffer wraps each time the number of samples exceeds MaxLogSamples.	No
NumParameters	The number of parameters from your Simulink model that you can tune or change.	No
NumSignals	The number of signals from your Simulink model that are available to be viewed with a scope.	No
OutputLog	Storage in the MATLAB workspace for the output or Y-vector logged during execution of the target application.	No
Parameters	List of tunable parameters. This list is visible only when ShowParameters is set to 'on': • Property value. Value of the parameter in a Simulink block. • Type. Data type of the parameter. Always double. • Size. Size of the parameter. For example, scalar, 1-by-2 vector, or 2-by-3 matrix. • Parameter name. Name of a parameter in a Simulink block. • Block name. Name of a Simulink block.	No

Property	Description	Writable
SampleTime	Time between samples. This value equals the step size, in seconds, for updating the model equations and posting the outputs. (See "User Interaction" for limitations on target property changes to sample times.)	Yes
Scopes	List of index numbers, with one index for each scope.	No
SessionTime	Time since the kernel started running on your target computer. This is also the elapsed time since you booted the target computer. Values are in seconds.	No
ShowParameters	Flag set to view or hide the list of parameters from your Simulink blocks. This list is shown when you display the properties for a target object. Values are 'on' and 'off'.	Yes
ShowSignals	Flag set to view or hide the list of signals from your Simulink blocks. This list is shown when you display the properties for a target object. Values are 'on' and 'off'.	Yes
Signals	List of viewable signals. This list is visible only when ShowSignals is set to 'on'. • Property name. S0, S1. . . • Property value. Value of the signal. • Block name. Name of the Simulink block the signal is from.	No
StateLog	Storage in the MATLAB workspace for the state or x-vector logged during execution of the target application.	No

Property	Description	Writable
Status	Execution status of your target application. Values are 'stopped' and 'running'.	No
StopTime	Time when the target application stops running. Values are in seconds. The original value is set in the **Solver** pane of the Configuration Parameters dialog box. When the ExecTime reaches StopTime, the application stops running.	Yes
TETLog	Storage in the MATLAB workspace for a vector containing task execution times during execution of the target application. To enable logging of the TET, you need to select the **Log Task Execution Time** check box in the **xPC Target options** pane of the Configuration Parameters dialog box.	No
TimeLog	Storage in the MATLAB workspace for the time or T-vector logged during execution of the target application.	No
ViewMode	Display either all scopes or a single scope on the target computer. Value is 'all' or a single scope index. This property is active only if the environment property TargetScope is set to enabled.	Yes

xpctarget.xpc

Purpose   Create target object representing target application

Syntax
```
target_object = xpctarget.xpc('mode', 'arg1', 'arg2')
target_object=xpctarget.xpc('target_object_name')
```

Arguments

target_object	Variable name to reference the target object
mode	Optionally, enter the communication mode

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

TCPIP	Enable TCP/IP connection with target computer.
RS232	Enable RS-232 connection with target computer.

arg1	Optionally, enter an argument based on the mode value:	
	IP address	If mode is 'TCPIP', enter the IP address of the target computer.
	COM port	If mode is 'RS232', enter the host COM port.
arg2	Optionally, enter an argument based on the mode value:	
	Port	If mode is 'TCPIP', enter the port number for the target computer.

	Baud rate	If mode is 'RS232', enter the baud rate for the connection between the host and target computer.
target_object_name		Target object name as specified in the xPC Target Explorer

Description Constructor of a target object (xpctarget.xpc Class). The target object represents the target application and target computer. You make changes to the target application by changing the target object using methods and properties.

If you have one target computer, or if you designate a target computer as the default one in your system, use the syntax

```
target_object=xpctarget.xpc
```

If you have multiple target computers in your system, use the following syntax to create the additional target objects.

```
target_object=xpctarget.xpc('mode', 'arg1', 'arg2')
```

If you have a target computer object in the xPC Target Explorer, you can use the following syntax to construct a corresponding target object from the MATLAB Command Window.

```
target_object=xpctarget.xpc('target_object_name')
```

Examples Before you build a target application, you can check the connection between your host and target computers by creating a target object, then using the xpctarget.xpc.targetping method to check the connection.

```
tg = xpctarget.xpc
 xPC Object
    Connected           = Yes
    Application         = loader

tg.targetping
```

```
ans =

success
```

If you have a second target computer for which you want to check the connection, create a second target object. In the following example, the connection with the second target computer is an RS-232 connection.

```
tg1=xpctarget.xpc('RS232','COM1','115200')

 xPC Object
   Connected            = Yes
   Application          = loader
```

If you have an xPC Target Explorer target object, and you want to construct a corresponding target object in the MATLAB Command Window, use a command like the following:

```
target_object=xpctarget.xpc('TargetPC1')
```

See Also   xpctarget.xpc.get (target application object) | xpctarget.xpc.set (target application object) | xpctarget.xpc.targetping

Purpose   Create scopes

Syntax   Create a scope and scope object without assigning to a MATLAB variable.

```
addscope(target_object, scope_type, scope_number)
target_object.addscope(scope_type, scope_number)
```

Create a scope, scope object, and assign to a MATLAB variable

```
scope_object = addscope(target_object,
    scope_type, scope_number)
scope_object = target_object.addscope(scope_type,
    scope_number)
```

Target computer command line — When you are using this command on the target computer, you can only add a target scope.

```
addscope
addscope scope_number
```

Arguments

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

Description   addscope creates a scope of the specified type and updates the target object property Scopes. This method returns a scope object vector. If the result is not assigned to a variable, the scope object properties are listed in the MATLAB window. The xPC Target product supports 10

target or host scopes, and eight file scopes, for a maximum of 28 scopes. If you try to add a scope with the same index as an existing scope, the result is an error.

Examples

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

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

or

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

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

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

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

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

Create a scope and scope object sc4 of type file using the method addscope. A file scope is created on the target computer with an index of 4. A scope object is created on the host computer and is assigned to the variable sc4. The target object property Scopes is changed from No scopes defined to 4.

```
sc4 = addscope(tg,'file',4) or sc4 = tg.addscope('file',4)
```

See Also   xpctarget.xpc.remscope | xpctarget.xpc.getscope

How To   • "Application and Driver Scripts"

xpctarget.xpc.close

Purpose   Close serial port connecting host computer with target computer

Syntax   close(target_object)
                target_object.close

Arguments   target_object   Name of a target object.

Description close closes the serial connection between the host computer and a
                target computer. If you want to use the serial port for another function
                without quitting the MATLAB window – for example, a modem – use
                this function to close the connection.

xpctarget.xpc.get (target application object)

Purpose   Return target application object property values

Syntax   `get(target_object, 'target_object_property')`

Arguments

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

Description   `get` gets the value of readable target object properties from a target object.

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

Property	Description	Writable
Application	Name of the Simulink model and target application built from that model.	No
AvgTET	Average task execution time. This value is an average of the measured CPU times, in seconds, to run the model equations and post outputs during each sample interval. Task execution time is nearly constant, with minor deviations due to cache, memory access, interrupt latency, and multirate model execution. The TET includes: • Complete I/O latency. • Data logging (the parts that happen in a real-time task). This includes data captured in scopes. • Asynchronous interruptions.	No

Property	Description	Writable
	• Parameter updating latency (if the **Double buffer parameter changes** parameter is set in the **xPC Target options** node of the Configuration Parameters dialog box). Note that the TET is not the only consideration in determining the minimum achievable sample time. Other considerations, not included in the TET, are: • Time required to measure TET • Interrupt latency required to schedule and run one step of the model	
CommunicationTimeOut	Communication timeout between host and target computer, in seconds.	Yes
Connected	Communication status between the host computer and the target computer. Values are 'Yes' and 'No'.	No
CPUoverload	CPU status for overload. If the target application requires more CPU time than the sample time of the model, this value is set from 'none' to 'detected' and the current run is stopped. Returning this status to 'none' requires either a faster processor or a larger sample time.	No
ExecTime	Execution time. Time, in seconds, since your target application started running. When the target application stops, the total execution time is displayed.	No

Property	Description	Writable
LogMode	Controls which data points are logged: • Time-equidistant logging. Logs a data point at every time interval. Set value to `'Normal'`. • 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.	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 **Signal Logging Buffer Size** by the number of logged signals. The **Signal Logging Buffer Size** box is in the **xPC Target options** pane of the Configuration Parameters dialog box.	No
MaxTET	Maximum task execution time. Corresponds to the slowest time (longest time measured), in seconds, to update model equations and post outputs.	No
MinTET	Minimum task execution time. Corresponds to the fastest time (smallest time measured), in seconds, to update model equations and post outputs.	No

Property	Description	Writable
Mode	Type of Simulink Coder code generation. Values are `'Real-Time Singletasking'`, `'Real-Time Multitasking'`, and `'Accelerate'`. The default value is `'Real-Time Singletasking'`. Even if you select `'Real-Time Multitasking'`, the actual mode can be `'Real-Time Singletasking'`. This happens if your model contains only one or two tasks and the sample rates are equal.	No
NumLogWraps	The number of times the circular buffer wrapped. The buffer wraps each time the number of samples exceeds `MaxLogSamples`.	No
NumParameters	The number of parameters from your Simulink model that you can tune or change.	No
NumSignals	The number of signals from your Simulink model that are available to be viewed with a scope.	No
OutputLog	Storage in the MATLAB workspace for the output or Y-vector logged during execution of the target application.	No

Property	Description	Writable
Parameters	List of tunable parameters. This list is visible only when ShowParameters is set to 'on': • Property value. Value of the parameter in a Simulink block. • Type. Data type of the parameter. Always double. • Size. Size of the parameter. For example, scalar, 1-by-2 vector, or 2-by-3 matrix. • Parameter name. Name of a parameter in a Simulink block. • Block name. Name of a Simulink block.	No
SampleTime	Time between samples. This value equals the step size, in seconds, for updating the model equations and posting the outputs. (See "User Interaction" for limitations on target property changes to sample times.)	Yes
Scopes	List of index numbers, with one index for each scope.	No
SessionTime	Time since the kernel started running on your target computer. This is also the elapsed time since you booted the target computer. Values are in seconds.	No
ShowParameters	Flag set to view or hide the list of parameters from your Simulink blocks. This list is shown when you display the properties for a target object. Values are 'on' and 'off'.	Yes

xpctarget.xpc.get (target application object)

Property	Description	Writable
ShowSignals	Flag set to view or hide the list of signals from your Simulink blocks. This list is shown when you display the properties for a target object. Values are `'on'` and `'off'`.	Yes
Signals	List of viewable signals. This list is visible only when ShowSignals is set to `'on'`. • Property name. S0, S1. . . • Property value. Value of the signal. • Block name. Name of the Simulink block the signal is from.	No
StateLog	Storage in the MATLAB workspace for the state or x-vector logged during execution of the target application.	No
Status	Execution status of your target application. Values are `'stopped'` and `'running'`.	No
StopTime	Time when the target application stops running. Values are in seconds. The original value is set in the **Solver** pane of the Configuration Parameters dialog box. When the ExecTime reaches StopTime, the application stops running.	Yes

Property	Description	Writable
TETLog	Storage in the MATLAB workspace for a vector containing task execution times during execution of the target application. To enable logging of the TET, you need to select the **Log Task Execution Time** check box in the **xPC Target options** pane of the Configuration Parameters dialog box.	No
TimeLog	Storage in the MATLAB workspace for the time or T-vector logged during execution of the target application.	No
ViewMode	Display either all scopes or a single scope on the target computer. Value is `'all'` or a single scope index. This property is active only if the environment property `TargetScope` is set to `enabled`.	Yes

Examples   List the value for the target object property `StopTime`. Notice that the property name is a string, in quotation marks, and not case sensitive.

```
get(tg,'stoptime') or tg.get('stoptime')
ans = 0.2
```

See Also   get | set | xpctarget.xpc.set (target application object) | xpctarget.xpcsc.get (scope object) | xpctarget.xpc.set (target application object)

xpctarget.xpc.getlog

Purpose	All or part of output logs from target object
Syntax	log = getlog(target_object, 'log_name', first_point, number_samples, decimation)

Arguments

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

Description

Use this function instead of the function get when you want only part of the data.

Examples

To get the first 1000 points in a log,

Out_log = getlog(tg, 'TETLog', 1, 1000)

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

Output_log = getlog(tg, 'TETLog', 1, 10, 2)
Time_log = getlog(tg, 'TimeLog', 1, 10, 2)
plot(Time_log, Output_log)

How To

- xpctarget.xpc.get (target application object)
- "Set Configuration Parameters"

Purpose	Value of target object parameter index

Syntax   getparam(target_object, parameter_index)

Arguments

target_object	Name of a target object. The default name is tg.
parameter_index	Index number of the parameter.

Description   getparam returns the value of the parameter associated with parameter_index.

Examples   Get the value of parameter index 5.

```
getparam(tg, 5)
ans = 400
```

xpctarget.xpc.getparamid

Purpose	Parameter index from parameter list

Syntax

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

Arguments

`target_object`	Name of a target object. The default name is `tg`.
`'block_name'`	Simulink block path without model name.
`'parameter_name'`	Name of a parameter within a Simulink block.

Description

getparamid returns the index of a parameter in the parameter list based on the path to the parameter name. The names must be entered in full and are case sensitive. Note, enter for block_name the mangled name that Simulink Coder uses for code generation.

Examples

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

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

Get the property index of a single block.

```
getparamid(tg, 'Gain1', 'Gain') ans = 5
```

See Also

xpctarget.xpc.getsignalid

How To

- "Application and Driver Scripts"
- "Why Does the getparamid Function Return Nothing?" on page 22-2

Purpose   Block path and parameter name from index list

Syntax   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
```

xpctarget.xpc.getscope

Purpose   Scope object pointing to scope defined in kernel

Syntax   scope_object_vector = getscope(target_object, scope_number)
                  scope_object = target_object.getscope(scope_number)

Arguments

target_object	Name of a target object.
scope_number_vector	Vector of existing scope indices listed in the target object property Scopes. The vector can have only one element.
scope_object	MATLAB variable for a new scope object vector. The vector can have only one scope object.

Description   getscope returns a scope object vector. If you try to get a nonexistent scope, the result is an error. You can retrieve the list of existing scopes using the method get(target_object, 'scopes') or target_object.scopes.

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

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

The following example gets the properties of all scopes on the target computer and creates a vector of scope objects on the host computer. If the target object has more than one scope, it create a vector of scope objects.

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

See Also   getxpcenv | xpctarget.xpc.remscope

How To   • "Application and Driver Scripts"

xpctarget.xpc.getsignal

Purpose   Value of target object signal index

Syntax   getsignal(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 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
```

Purpose	Signal index or signal property from signal list

Syntax

```
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, the xPC Target software 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. Note, enter for block_name the mangled name that Simulink Coder uses for code generation.

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

xpctarget.xpc.getparamid

How To

- "Application and Driver Scripts"

- "Why Does the getparamid Function Return Nothing?" on page 22-2

xpctarget.xpc.getsignalidsfromlabel

Purpose	Return vector of signal indices

Syntax

```
getsignalidsfromlabel(target_object, signal_label)
target_object.getsignalidsfromlabel(signal_label)
```

Arguments

target_object	Name of a target object. The default name is tg.
signal_label	Signal label (from Simulink model).

Description   getsignalidsfromlabel returns a vector of one or more signal indices that are associated with the labeled signal, signal_label. This function assumes that you have labeled the signal for which you request the index (see the **Signal name** parameter of the "Signal Properties Controls"). Note that the xPC Target software refers to Simulink signal names as signal labels.

Examples   Get the vector of signal indices for a signal labeled Gain.

```
>> tg.getsignalidsfromlabel('xpcoscGain')
ans =
0
```

See Also   xpctarget.xpc.getsignallabel

| **Purpose** | Return signal label |

Syntax   getsignallabel(target_object, signal_index)
              target_object.getsignallabel(signal_index)

Arguments

| target_object | Name of a target object. The default name is tg. |
| signal_index | Index number of the signal. |

Description   getsignallabel returns the signal label for the specified signal index, signal_index. signal_label. This function assumes that you have labeled the signal for which you request the label (see the **Signal name** parameter of the "Signal Properties Controls"). Note that the xPC Target software refers to Simulink signal names as signal labels.

Examples   >> getsignallabel(tg, 0)
                ans =
                xpcoscGain

See Also   xpctarget.xpc.getsignalidsfromlabel

xpctarget.xpc.getsignalname

| **Purpose** | Signal name from index list |

Syntax

```
getsignalname(target_object, signal_index)
target_object.getsignalname(signal_index)
```

Arguments

| target_object | Name of a target object. The default name is tg. |
| signal_index | Index number of the signal. |

Description

getparamname returns one argument string, signal name, from the index list for the specified signal index.

Examples

Get the signal name of signal ID 2.

```
[sigName]=getsignalname(tg,2)
sigName =
Gain2
```

Purpose   Determine which PCI boards are installed in target computer

Syntax   getxpcpci(target_object, 'type_of_boards')
                  getxpcpci(target_object, 'verbose')

Arguments

target_object	Variable name to reference the target object.
type_of_boards	Values are no arguments, 'all', and 'supported'.
verbose	Argument to include the base address register information in the PCI device display.

Description   The getxpcpci function displays, in the MATLAB window, which PCI
                  boards are installed in the target computer. By default, getxpcpci
                  displays this information for the target object, tg. If you have multiple
                  target computers in your system, you can call the getxpcpci function
                  for a particular target object, target_object.

                  Only devices supported by driver blocks in the xPC Target block library
                  are displayed. The information includes the PCI bus number, slot
                  number, assigned IRQ number, manufacturer name, board name, device
                  type, manufacturer PCI ID, base address, and the board PCI ID itself.

                  The following preconditions must be met before you can use this
                  function:

                  • The host-target communication link must be working. (The function
                    xpctargetping must return success before you can use the function
                    getxpcpci.)

                  • Either a target application is loaded or the loader is active. The
                    latter is used to query for resources assigned to a specific PCI device,
                    which have to be provided to a driver block dialog box before the
                    model build process.

Examples   The following example displays the installed PCI devices, not only the devices supported by the xPC Target block library. This includes graphics controllers, network cards, SCSI cards, and even devices that are part of the motherboard chip set (for example, PCI-to-PCI bridges).

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

The following example displays a list of the currently supported PCI devices in the xPC Target block library, including subvendor and subdevice information.

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

The following example displays a list of the currently supported PCI devices in the xPC Target block library, including subvendor and subdevice information and base address register contents.

```
getxpcpci('verbose')
```

When called with the 'supported' option, getxpcpci does not access the target computer.

To display the list of PCI devices installed on the target computer, tg1, first create a target object, tg1, for that target computer. Then, call getxpcpci with the 'all' option. For example:

```
tg1=xpctarget.xpc('RS232','COM1','115200')
getxpcpci(tg1, 'all')
```

To return the result of a getxpcpci query in the struct pcidevs instead of displaying it, assign the function to pcidevs. The struct pcidevs is an array with one element for each detected PCI device. Each element combines the information by a set of field names. The struct contains more information compared to the displayed list. Its contents vary according to the options you specify for the function.

```
pcidevs = getxpcpci
```

Purpose	Download target application to target computer

Syntax

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

Arguments

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

Description

Before using this function, the target computer must be booted with the xPC Target kernel, and the target application must be built in the current working folder on the host computer.

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

Note If you are running in Standalone mode, this command has no effect. To load a new application, you must rebuild the standalone application with the new application, then reboot the target computer with the updated standalone application.

Examples

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

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

See Also xpctarget.xpc.unload

How To • "Application and Driver Scripts"

xpctarget.xpc.loadparamset

Purpose	Restore parameter values saved in specified file

Syntax

```
loadparamset(target_object,'filename')
target_object.loadparamset('filename')
```

Arguments

target_object	Name of an existing target object.
filename	Enter the name of the file that contains the saved parameters.

Description

loadparamset restores the target application parameter values saved in the file filename. This file must be located on a local drive of the target computer. This method assumes that you have a parameter file from a previous run of the xpctarget.xpc.saveparamset method.

See Also

xpctarget.xpc.saveparamset

Purpose   Reboot target computer

Syntax   **MATLAB command line**

reboot(target_object)

Target computer command line

reboot

Arguments   target_object   Name of an existing target object.

Description   reboot reboots the target computer, and if a target boot disk is still present, the xPC Target kernel is reloaded.

On the target computer command line, you can use the corresponding command reboot.

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

Note This method might not work on some target hardware.

See Also   xpctarget.xpc.load | xpctarget.xpc.unload

xpctarget.xpc.remscope

Purpose   Remove scope from target computer

Syntax   **MATLAB command line**

```
remscope(target_object, scope_number_vector)
target_object.remscope(scope_number_vector)
remscope(target_object)
target_object.remscope
```

Target computer command line

```
remscope scope_number
remscope 'all'
```

Arguments

target_object	Name of a target object. The default name is tg.
scope_number_vector	Vector of existing scope indices listed in the target object property Scopes.
scope_number	Single scope index.

Description   If a scope index is not given, the method remscope deletes all scopes on the target computer. The method remscope has no return value. The scope object representing the scope on the host computer is not deleted.

Note that you can only permanently remove scopes that are added with the method addscope. This is a scope that is outside a model. If you remove a scope that has been added through a scope block (the scope block is inside the model), a subsequent run of that model creates the scope again.

Examples   Remove a single scope.

```
remscope(tg,1)
```

or

```
tg.remscope(1)
```

Remove two scopes.

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

or

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

Remove all scopes.

```
remscope(tg)
```

or

```
tg.remscope
```

See Also   xpctarget.xpc.addscope | xpctarget.xpc.getscope

How To   • "Application and Driver Scripts"

xpctarget.xpc.saveparamset

Purpose   Save current target application parameter values

Syntax   saveparamset(target_object,'filename')
                   target_object.saveparamset('filename')

Arguments

target_object	Name of an existing target object.
filename	Enter the name of the file to contain the saved parameters.

Description   saveparamset saves the target application parameter values in the
                   file filename. This method saves the file on a local drive of the target
                   computer (C:\ by default). You can later reload these parameters with
                   the xpctarget.xpc.loadparamset function.

                   You might want to save target application parameter values if you
                   change these parameter values while the application is running in
                   real time. Saving these values enables you to easily recreate target
                   application parameter values from a number of application runs.

See Also   xpctarget.xpc.loadparamset

xpctarget.xpc.set (target application object)

Purpose   Change target application object property values

Syntax   **MATLAB command line**

```
set(target_object)
set(target_object, 'property_name1', 'property_value1',
'property_name2', 'property_value2', . . .)
target_object.set('property_name1', 'property_value1')
set(target_object, property_name_vector,
property_value_vector)
target_object.property_name = property_value
```

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

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

Arguments | target_object | Name of a target object. |
|------------------|--------------------------|
| 'property_name' | Name of a target object property. Always use quotation marks. |
| property_value | Value for a target object property. Always use quotation marks for character strings; quotation marks are optional for numbers. |

Description   set sets the properties of the target object. Not all properties are user
writable.

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

are listed in the following table. This table includes a description of the properties:

Property	Description	Writable
CommunicationTimeOut	Communication timeout between host and target computer, in seconds.	Yes
LogMode	Controls which data points are logged: • Time-equidistant logging. Logs a data point at every time interval. Set value to 'Normal'. • 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.	Yes
SampleTime	Time between samples. This value equals the step size, in seconds, for updating the model equations and posting the outputs. See "User Interaction" for limitations on target property changes to sample times.	Yes

Property	Description	Writable
ShowParameters	Flag set to view or hide the list of parameters from your Simulink blocks. This list is shown when you display the properties for a target object. Values are 'on' and 'off'.	Yes
ShowSignals	Flag set to view or hide the list of signals from your Simulink blocks. This list is shown when you display the properties for a target object. Values are 'on' and 'off'.	Yes
StopTime	Time when the target application stops running. Values are in seconds. The original value is set in the **Solver** pane of the Configuration Parameters dialog box. When the ExecTime reaches StopTime, the application stops running.	Yes
ViewMode	Display either all scopes or a single scope on the target computer. Value is 'all' or a single scope index. This property is active only if the environment property TargetScope is set to enabled.	Yes

xpctarget.xpc.set (target application object)

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

Examples

Get a list of writable properties for a scope object.

```
set(tg)
ans =
           StopTime: {}
         SampleTime: {}
           ViewMode: {}
            LogMode: {}
     ShowParameters: {}
         ShowSignals: {}
```

Change the property `ShowSignals` to on.

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

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

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

See Also

get | set | xpctarget.xpc.get (target application object) | xpctarget.xpcsc.get (scope object) | xpctarget.xpcsc.set (scope object)

How To

• "Application and Driver Scripts"

Purpose   Change writable target object parameters

Syntax   setparam(target_object, parameter_index, parameter_value)

Arguments

target_object	Name of an existing target object. The default name is tg.
parameter_index	Index number of the parameter.
parameter_value	Value for a target object parameter.

Description   Method of a target object. Set the value of the target parameter. This method returns a structure that stores the parameter index, previous parameter values, and new parameter values in the following fields:

- parIndexVec

- OldValues

- NewValues

Examples   Set the value of parameter index 5 to 100.

```
setparam(tg, 5, 100)
ans =
parIndexVec: 5
OldValues: 400
NewValues: 100
```

Simultaneously set values for multiple parameters. Use the cell array format to specify new parameter values.

```
setparam(tg, [1 5],{10,100})
ans =
parIndexVec: [1 5]
OldValues: {[2]  [4]}
NewValues: {[10]  [100]}
```

xpctarget.xpc.start (target application object)

Purpose   Start execution of target application on target computer

Syntax   **MATLAB command line**

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

Target computer command line

```
start
```

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

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

Examples   Start the target application represented by the target object tg.

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

See Also   xpctarget.xpc.stop (target application object)
                   | xpctarget.xpc.load | xpctarget.xpc.unload |
                   xpctarget.xpcsc.stop (scope object)

xpctarget.xpc.stop (target application object)

Purpose   Stop execution of target application on target computer

Syntax   **MATLAB command line**

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

Target computer command line

```
stop
```

Arguments   target_object   Name of a target object.

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

Examples   Stop the target application represented by the target object tg.

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

See Also   xpctarget.xpc.start (target application object) |
                     xpctarget.xpcsc.stop (scope object) | xpctarget.xpcsc.start
                     (scope object)

xpctarget.xpc.targetping

Purpose	Test communication between host and target computers
Syntax	`targetping(target_object)` `target_object.targetping`
Arguments	`target_object` Name of a target object.
Description	Method of a target object. Use this method to ping a target computer from the host computer. This method returns `success` if the xPC Target kernel is loaded and running and communication is working between host and target, otherwise it returns `failed`. This function works with both RS-232 and TCP/IP communication.

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

Examples	Ping the communication between the host and the target object `tg`. `targetping(tg)` or `tg.targetping`
See Also	`xpctarget.xpc`

Purpose   Remove current target application from target computer

Syntax   unload(target_object)
                   target_object.unload

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

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

Note If you are running in StandAlone mode, this command has no effect. To unload and reload a new application, you must rebuild the standalone application with the new application, then reboot the target computer with the updated standalone application.

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

                   unload(tg) or tg.unload

See Also   xpctarget.xpc.load | xpctarget.xpc.reboot

xpctarget.xpcfs Class

Purpose   Control and access properties of file scopes

Description   The scope gets a data package from the kernel and stores the data in a file in the target computer file system. Depending on the setting of `WriteMode`, the file size is or is not continuously updated. You can then transfer the data to another computer for examination or plotting.

Methods

These methods are inherited from `xpctarget.xpcsc Class`.

Method	Description
`xpctarget.xpcsc.addsignal`	Add signals to scope represented by scope object
`xpctarget.xpcsc.get` (scope object)	Return property values for scope objects
`xpctarget.xpcsc.remsignal`	Remove signals from scope represented by scope object
`xpctarget.xpcsc.set` (scope object)	Change property values for scope objects
`xpctarget.xpcsc.start` (scope object)	Start execution of scope on target computer
`xpctarget.xpcsc.stop` (scope object)	Stop execution of scope on target computer
`xpctarget.xpcsc.trigger`	Software-trigger start of data acquisition for scope(s)

Properties

These properties are inherited from `xpctarget.xpcsc Class`.

Property	Description	Writable
`Application`	Name of the Simulink model associated with this scope object.	No
`Decimation`	A number `n`, where every `n`th sample is acquired in a scope window.	Yes

Property	Description	Writable
NumPrePostSamples	Number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set TriggerMode to 'FreeRun', this property has no effect on data acquisition.	Yes
NumSamples	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. For file scopes, this parameter works in conjunction with the **AutoRestart** check box. If the **AutoRestart** box is selected, the file scope collects data up to **Number of Samples**, then starts over again, overwriting the buffer. If the **AutoRestart** box is not selected, the file scope collects data only up to **Number of Samples**, then stops.	Yes
ScopeId	A numeric index, unique for each scope.	No
Signals	List of signal indices from the target object to display on the scope.	Yes
Status	Indicate whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are 'Acquiring', 'Ready for being Triggered', 'Interrupted', and 'Finished'.	No

xpctarget.xpcfs Class

Property	Description	Writable
TriggerLevel	If TriggerMode is 'Signal', indicates the value the signal has to cross to trigger the scope and start acquiring data. The trigger level can be crossed with either a rising or falling signal.	Yes
TriggerMode	Trigger mode for a scope. Valid values are 'FreeRun' (default), 'Software', 'Signal', and 'Scope'.	Yes
TriggerSample	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. 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.	Yes
TriggerScope	If TriggerMode is 'Scope', identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered. You do this by setting the slave scope property TriggerScope to the scope index of the master scope.	Yes

Property	Description	Writable
TriggerSignal	If `TriggerMode` is `'Signal'`, identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property `Signal`.	Yes
TriggerSlope	If `TriggerMode` is `'Signal'`, indicates whether the trigger is on a rising or falling signal. Values are `'Either'` (default), `'Rising'`, and `'Falling'`.	Yes
Type	Determines whether the scope is displayed on the host computer or on the target computer. Values are `'Host'`, `'Target'`, and `'File'`.	Yes

These properties are specific to class `xpcfs`.

Property	Description	Writeable
AutoRestart	Values are `'on'` and `'off'`. For file scopes, enable the file scope to collect data up to the number of samples (`NumSamples`), then start over again, appending the new data to the end of the signal data file. Clear the **AutoRestart** check box to have the file scope collect data up to **Number of samples**, then stop. If the named signal data file already exists when you start the target application, the software overwrites the old data with the new signal data. To use the `DynamicFileName` property, set `AutoRestart` to `'on'` first.	No

Property	Description	Writeable
	For host or target scopes, this parameter has no effect.	
DynamicFileName	Values are `'on'` and `'off'`. By default, the value is `'off'`.	Yes
	Enable the ability to dynamically create multiple log files for file scopes.	
	To use `DynamicFileName`, set `AutoRestart` to `'on'` first. When you enable `DynamicFileName`, configure `Filename` to create incrementally numbered file names for the multiple log files. Failure to do so causes an error when you try to start the scope.	
	You can enable the creation of up to 99999999 files (`<%%%%%%%%>.dat`). The length of a file name, including the specifier, cannot exceed eight characters.	
	For host or target scopes, this parameter has no effect.	

Property	Description	Writeable

Filename	Provide a name for the file to contain the signal data. By default, the target computer writes the signal data to a file named `C:\data.dat` for scope blocks. Note that for file scopes created through the MATLAB interface, no name is initially assigned to `FileName`. After you start the scope, the software assigns a name for the file to acquire the signal data. This name typically consists of the scope object name, `ScopeId`, and the beginning letters of the first signal added to the scope.
	
	If you set `DynamicFileName` and `AutoRestart` to `'on'`, configure `Filename` to dynamically increment. Use a base file name, an underscore (_), and a < > specifier. Within the specifier, enter one to eight % symbols. Each symbol % represents a decimal location in the file name. The specifier can appear anywhere in the file name. For example, the following value for `Filename`, `C:\work\file_<%%%>.dat` creates file names with the following pattern:
	
	`file_001.dat`
	`file_002.dat`
	`file_003.dat`
		No

Property	Description	Writeable
	The last file name of this series will be file_999.dat. If the function is still logging data when the last file name reaches its maximum size, the function starts from the beginning and overwrites the first file name in the series. If you do not retrieve the data from existing files before they are overwritten, the data is lost. For host or target scopes, this parameter has no effect.	
MaxWriteFileSize	Provide the maximum size of Filename, in bytes. This value must be a multiple of WriteSize. Default is 536870912. When the size of a log file reaches MaxWriteFileSize, the software creates a subsequently numbered file name, and continues logging data to that file, up until the highest log file number you have specified. If the software cannot create additional log files, it overwrites the first log file. For host or target scopes, this parameter has no effect.	Yes

Property	Description	Writeable
Mode	**Note** The Mode property will be removed in a future release. • For target scopes, use DisplayMode. • For file scopes, use WriteMode. • For host scopes, this parameter has no effect.	Yes
WriteMode	For file scopes, specify when a file allocation table (FAT) entry is updated. Values are 'Lazy' or 'Commit'. Both modes write the signal data to the file. With 'Commit' mode, each file write operation simultaneously updates the FAT entry for the file. This mode is slower, but the file system maintains 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). For host or target scopes, this parameter has no effect.	Yes
WriteSize	Enter the block size, in bytes, of the data chunks. This parameter	Yes

Property	Description	Writeable
	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 better performance.	
	If you experience a system crash, you can expect to lose an amount of data the size of WriteSize.	
	For host or target scopes, this parameter has no effect.	

Purpose   Add signals to scope represented by scope object

Syntax   **MATLAB command line**

```
addsignal(scope_object_vector, signal_index_vector)
scope_object_vector.addsignal(signal_index_vector)
```

Target command line

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

Arguments

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

Description   addsignal adds signals to a scope object. The signals must be specified by their indices, which you can retrieve using the target object method getsignalid. If the scope_object_vector has two or more scope objects, the same signals are assigned to each scope.

Note   You must stop the scope before you can add a signal to it.

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

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

xpctarget.xpcsc.addsignal

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

```
sc1.Signals
Signals               = 1  : Signal Generator
                        O  : Integrator1
```

Another way to add signals without using the method `addsignal` is to use the scope object method `set`.

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

Or, to directly assign signal values to the scope object property `Signals`,

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

See Also   xpctarget.xpcsc.remsignal | xpctarget.xpcsc.set (scope object) | xpctarget.xpc.addscope | xpctarget.xpc.getsignalid

xpctarget.xpcsc.get (scope object)

Purpose   Return property values for scope objects

Syntax   get(scope_object_vector)
                 get(scope_object_vector, 'scope_object_property')
                 get(scope_object_vector, scope_object_property_vector)

Arguments

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

Description get gets the value of readable scope object properties from a scope object or the same property from each scope object in a vector of scope objects. Scope object properties let you select signals to acquire, set triggering modes, and access signal information from the target application. You can view and change these properties using scope object methods.

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

Property	Description	Writable
Application	Name of the Simulink model associated with this scope object.	No
AutoRestart	Values are 'on' and 'off'. For file scopes, enable the file scope to collect data up to the number of samples (NumSamples), then start over again, appending the new data to the end of the signal data file. Clear the **AutoRestart** check box to have the file scope collect data up to **Number of samples**, then stop.	No

Property	Description	Writable
	If the named signal data file already exists when you start the target application, the software overwrites the old data with the new signal data.	
	For host or target scopes, this parameter has no effect.	
	To use the `DynamicFileName` property, set `AutoRestart` to `'on'` first.	
Data	Contains the output data for a single data package from a scope.	No
	For target or file scopes, this parameter has no effect.	
Decimation	A number n, where every nth sample is acquired in a scope window.	Yes
DisplayMode	For target scopes, indicate how a scope displays the signals. Values are `'Numerical'`, `'Redraw'` (default), `'Sliding'`, and `'Rolling'`.	Yes
	For host or file scopes, this parameter has no effect. .	
DynamicFileName	Values are `'on'` and `'off'`. By default, the value is `'off'`.	Yes
	Enable the ability to dynamically create multiple log files for file scopes.	
	To use `DynamicFileName`, set `AutoRestart` to `'on'` first. When you enable `DynamicFileName`, configure `Filename` to create incrementally numbered file names for the multiple log files. Failure to do so causes an error when you try to start the scope.	

Property	Description	Writable
	You can enable the creation of up to 99999999 files (`<%%%%%%%%>.dat`). The length of a file name, including the specifier, cannot exceed eight characters. For host or file scopes, this parameter has no effect.	

Filename	Provide a name for the file to contain the signal data. By default, the target computer writes the signal data to a file named `C:\data.dat` for scope blocks. Note that for file scopes created through the MATLAB interface, no name is initially assigned to `FileName`. After you start the scope, the software assigns a name for the file to acquire the signal data. This name typically consists of the scope object name, `ScopeId`, and the beginning letters of the first signal added to the scope.
	
	If you set `DynamicFileName` and `AutoRestart` to `'on'`, configure `Filename` to dynamically increment. Use a base file name, an underscore (`_`), and a `< >` specifier. Within the specifier, enter one to eight `%` symbols. Each symbol `%` represents a decimal location in the file name. The specifier can appear anywhere in the file name. For example, the following value for `Filename`, `C:\work\file_<%%%>.dat` creates file names with the following pattern:
	
	`file_001.dat`
	`file_002.dat`
	`file_003.dat`
	
	The last file name of this series will be `file_999.dat`. If the function is still logging data when the last file name reaches its maximum
		No

Property	Description	Writable
	size, the function starts from the beginning and overwrites the first file name in the series. If you do not retrieve the data from existing files before they are overwritten, the data is lost. For host or target scopes, this parameter has no effect.	
MaxWriteFileSize	Provide the maximum size of `Filename`, in bytes. This value must be a multiple of `WriteSize`. Default is `536870912`. When the size of a log file reaches `MaxWriteFileSize`, the software creates a subsequently numbered file name, and continues logging data to that file, up until the highest log file number you have specified. If the software cannot create additional log files, it overwrites the first log file.	Yes
Grid	Values are `'on'` and `'off'`. For host or file scopes, this parameter has no effect.	Yes
Mode	**Note** The `Mode` property will be removed in a future release. • For target scopes, use `DisplayMode`. • For file scopes, use `WriteMode`. • For host scopes, this parameter has no effect.	Yes

Property	Description	Writable
NumPrePostSamples	Number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set `TriggerMode` to `'FreeRun'`, this property has no effect on data acquisition.	Yes
NumSamples	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. For file scopes, this parameter works in conjunction with the **AutoRestart** check box. If the **AutoRestart** box is selected, the file scope collects data up to **Number of Samples**, then starts over again, overwriting the buffer. If the **AutoRestart** box is not selected, the file scope collects data only up to **Number of Samples**, then stops.	Yes
ScopeId	A numeric index, unique for each scope.	No
Signals	List of signal indices from the target object to display on the scope.	Yes
Status	Indicate whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are `'Acquiring'`, `'Ready for being Triggered'`, `'Interrupted'`, and `'Finished'`.	No

Property	Description	Writable
Time	Contains the time data for a single data package from a scope.	No
TriggerLevel	If TriggerMode is 'Signal', indicates the value the signal has to cross to trigger the scope and start acquiring data. The trigger level can be crossed with either a rising or falling signal.	Yes
TriggerMode	Trigger mode for a scope. Valid values are 'FreeRun' (default), 'Software', 'Signal', and 'Scope'.	Yes
TriggerSample	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. 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.	Yes
TriggerScope	If TriggerMode is 'Scope', identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered. You do this by setting the slave scope property TriggerScope to the scope index of the master scope.	Yes

Property	Description	Writable
TriggerSignal	If TriggerMode is 'Signal', identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property Signal.	Yes
TriggerSlope	If TriggerMode is 'Signal', indicates whether the trigger is on a rising or falling signal. Values are 'Either' (default), 'Rising', and 'Falling'.	Yes
Type	Determines whether the scope is displayed on the host computer or on the target computer. Values are 'Host', 'Target', and 'File'.	Yes
WriteMode	For file scopes, specify when a file allocation table (FAT) entry is updated. Values are 'Lazy' or 'Commit'. Both modes write the signal data to the file. With 'Commit' mode, each file write operation simultaneously updates the FAT entry for the file. This mode is slower, but the file system maintains 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). For host or target scopes, this parameter has no effect.	Yes

xpctarget.xpcsc.get (scope object)

Property	Description	Writable
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 better performance. If you experience a system crash, you can expect to lose an amount of data the size of WriteSize. For host or target scopes, this parameter has no effect.	Yes
YLimit	Minimum and maximum *y*-axis values. This property can be set to 'auto'. For host or file scopes, this parameter has no effect.	Yes

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

```
get(sc)
```

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

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

See Also   get | set | xpctarget.xpcsc.set (scope object) | xpctarget.xpc.set (target application object)

Purpose   Base class for the scope classes

Description   This is the base class for the scope classes, xpctarget.xpcfs Class, xpctarget.xpcschost Class, and xpctarget.xpcsctg Class. All methods and properties are inherited by the derived classes. When a mixture of derived classes are stored in a scope collection, only the base class methods and properties are available. The scope class constructors are Private and are not intended to be called from the MATLAB prompt.

A scope acquires data from the target application and displays that data on the target computer, uploads the data to the host computer, or stores that data in a file in the target computer file system. The target, host, or file scopes run on the target computer.

Methods

These methods are inherited by the derived classes.

Method	Description
xpctarget.xpcsc.addsignal	Add signals to scope represented by scope object
xpctarget.xpcsc.get (scope object)	Return property values for scope objects
xpctarget.xpcsc.remsignal	Remove signals from scope represented by scope object
xpctarget.xpcsc.set (scope object)	Change property values for scope objects
xpctarget.xpcsc.start (scope object)	Start execution of scope on target computer
xpctarget.xpcsc.stop (scope object)	Stop execution of scope on target computer
xpctarget.xpcsc.trigger	Software trigger start of data acquisition for scope(s)

Properties

These properties are inherited by the derived classes.

xpctarget.xpcsc Class

Property	Description	Writable
Application	Name of the Simulink model associated with this scope object.	No
Decimation	A number n, where every nth sample is acquired in a scope window.	Yes
NumPrePostSamples	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.	Yes
NumSamples	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. For file scopes, this parameter works in conjunction with the **AutoRestart** check box. If the **AutoRestart** box is selected, the file scope collects data up to **Number of Samples**, then starts over again, overwriting the buffer. If the **AutoRestart** box is not selected, the file scope collects data only up to **Number of Samples**, then stops.	Yes
ScopeId	A numeric index, unique for each scope.	No
Signals	List of signal indices from the target object to display on the scope.	Yes

Property	Description	Writable
Status	Indicate whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are `'Acquiring'`, `'Ready for being Triggered'`, `'Interrupted'`, and `'Finished'`.	No
TriggerLevel	If `TriggerMode` is `'Signal'`, indicates the value the signal has to cross to trigger the scope and start acquiring data. The trigger level can be crossed with either a rising or falling signal.	Yes
TriggerMode	Trigger mode for a scope. Valid values are `'FreeRun'` (default), `'Software'`, `'Signal'`, and `'Scope'`.	Yes
TriggerSample	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. 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.	Yes

xpctarget.xpcsc Class

Property	Description	Writable
TriggerScope	If `TriggerMode` is `'Scope'`, identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered. You do this by setting the slave scope property `TriggerScope` to the scope index of the master scope.	Yes
TriggerSignal	If `TriggerMode` is `'Signal'`, identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property `Signal`.	Yes
TriggerSlope	If `TriggerMode` is `'Signal'`, indicates whether the trigger is on a rising or falling signal. Values are `'Either'` (default), `'Rising'`, and `'Falling'`.	Yes
Type	Determines whether the scope is displayed on the host computer or on the target computer. Values are `'Host'`, `'Target'`, and `'File'`.	Yes

Purpose   Remove signals from scope represented by scope object

Syntax   **MATLAB command line**

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

Target command line

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

Arguments

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

Description   remsignal removes signals from a scope object. The signals must be specified by their indices, which you can retrieve using the target object method getsignalid. If the scope_index_vector has two or more scope objects, the same signals are removed from each scope. The argument signal_index is optional; if it is left out, all signals are removed.

Note You must stop the scope before you can remove a signal from it.

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

```
sc1.get('signals')
ans= 0 1
```

xpctarget.xpcsc.remsignal

Remove signals from the scope on the target computer with the scope object property Signals updated.

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

or

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

See Also   xpctarget.xpcsc.remsignal | xpctarget.xpc.getsignalid

Purpose   Change property values for scope objects

Syntax
```
set(scope_object_vector)
set(scope_object_vector, property_name1, property_value1,
property_name2, property_value2, . . .)
scope_object_vector.set('property_name1', property_value1,
. . .)
set(scope_object, 'property_name', property_value, . . .)
```

Arguments

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.

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.

xpctarget.xpcsc.set (scope object)

Property	Description	Writable
Application	Name of the Simulink model associated with this scope object.	No
AutoRestart	Values are `'on'` and `'off'`. For file scopes, enable the file scope to collect data up to the number of samples (`NumSamples`), then start over again, appending the new data to the end of the signal data file. Clear the **AutoRestart** check box to have the file scope collect data up to **Number of samples**, then stop. If the named signal data file already exists when you start the target application, the software overwrites the old data with the new signal data. For host or target scopes, this parameter has no effect. To use the `DynamicFileName` property, set `AutoRestart` to `'on'` first.	No
Data	Contains the output data for a single data package from a scope. For target or file scopes, this parameter has no effect.	No
Decimation	A number n, where every nth sample is acquired in a scope window.	Yes
DisplayMode	For target scopes, indicate how a scope displays the signals. Values are `'Numerical'`, `'Redraw'` (default), `'Sliding'`, and `'Rolling'`. For host or file scopes, this parameter has no effect. .	Yes

Property	Description	Writable
DynamicFileName	Values are `'on'` and `'off'`. By default, the value is `'off'`. Enable the ability to dynamically create multiple log files for file scopes. To use DynamicFileName, set AutoRestart to `'on'` first. When you enable DynamicFileName, configure Filename to create incrementally numbered file names for the multiple log files. Failure to do so causes an error when you try to start the scope. You can enable the creation of up to 99999999 files (`<%%%%%%%%>.dat`). The length of a file name, including the specifier, cannot exceed eight characters. For host or file scopes, this parameter has no effect.	Yes
Filename	Provide a name for the file to contain the signal data. By default, the target computer writes the signal data to a file named `C:\data.dat` for scope blocks. Note that for file scopes created through the MATLAB interface, no name is initially assigned to FileName. After you start the scope, the software assigns a name for the file to acquire the signal data. This name typically consists of the scope object name, ScopeId, and the beginning letters of the first signal added to the scope. If you set DynamicFileName and AutoRestart to `'on'`, configure Filename to dynamically increment. Use a base file name, an underscore (_), and a `< >` specifier. Within the specifier, enter one to eight % symbols. Each symbol % represents a decimal location in the file name.	No

Property	Description	Writable

	The specifier can appear anywhere in the file name. For example, the following value for `Filename`, `C:\work\file_<%%%>.dat` creates file names with the following pattern:
	
	`file_001.dat`
	`file_002.dat`
	`file_003.dat`
	
	The last file name of this series will be `file_999.dat`. If the function is still logging data when the last file name reaches its maximum size, the function starts from the beginning and overwrites the first file name in the series. If you do not retrieve the data from existing files before they are overwritten, the data is lost.
	
	For host or target scopes, this parameter has no effect.	

MaxWriteFileSize	Provide the maximum size of `Filename`, in bytes. This value must be a multiple of `WriteSize`. Default is `536870912`. When the size of a log file reaches `MaxWriteFileSize`, the software creates a subsequently numbered file name, and continues logging data to that file, up until the highest log file number you have specified. If the software cannot create additional log files, it overwrites the first log file.	Yes
Grid	Values are `'on'` and `'off'`. For host or file scopes, this parameter has no effect.	Yes

Property	Description	Writable
Mode	**Note** The Mode property will be removed in a future release. • For target scopes, use DisplayMode. • For file scopes, use WriteMode. • For host scopes, this parameter has no effect.	Yes
NumPrePostSamples	Number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set TriggerMode to 'FreeRun', this property has no effect on data acquisition.	Yes
NumSamples	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. For file scopes, this parameter works in conjunction with the **AutoRestart** check box. If the **AutoRestart** box is selected, the file scope collects data up to **Number of Samples**, then starts over again, overwriting the buffer. If the **AutoRestart** box is not selected, the file scope collects data only up to **Number of Samples**, then stops.	Yes

xpctarget.xpcsc.set (scope object)

Property	Description	Writable
ScopeId	A numeric index, unique for each scope.	No
Signals	List of signal indices from the target object to display on the scope.	Yes
Status	Indicate whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are `'Acquiring'`, `'Ready for being Triggered'`, `'Interrupted'`, and `'Finished'`.	No
Time	Contains the time data for a single data package from a scope.	No
TriggerLevel	If `TriggerMode` is `'Signal'`, indicates the value the signal has to cross to trigger the scope and start acquiring data. The trigger level can be crossed with either a rising or falling signal.	Yes
TriggerMode	Trigger mode for a scope. Valid values are `'FreeRun'` (default), `'Software'`, `'Signal'`, and `'Scope'`.	Yes
TriggerSample	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. 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	Yes

Property	Description	Writable
	acquired one sample after the last sample of the triggering scope.	
TriggerScope	If `TriggerMode` is `'Scope'`, identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered. You do this by setting the slave scope property `TriggerScope` to the scope index of the master scope.	Yes
TriggerSignal	If `TriggerMode` is `'Signal'`, identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property `Signal`.	Yes
TriggerSlope	If `TriggerMode` is `'Signal'`, indicates whether the trigger is on a rising or falling signal. Values are `'Either'` (default), `'Rising'`, and `'Falling'`.	Yes
Type	Determines whether the scope is displayed on the host computer or on the target computer. Values are `'Host'`, `'Target'`, and `'File'`.	Yes
WriteMode	For file scopes, specify when a file allocation table (FAT) entry is updated. Values are `'Lazy'` or `'Commit'`. Both modes write the signal data to the file. With `'Commit'` mode, each file write operation simultaneously updates the FAT entry for the file. This mode is slower, but the file system maintains 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). For host or target scopes, this parameter has no effect. .	Yes

xpctarget.xpcsc.set (scope object)

Property	Description	Writable
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 better performance. If you experience a system crash, you can expect to lose an amount of data the size of WriteSize. For host or target scopes, this parameter has no effect.	Yes
YLimit	Minimum and maximum *y*-axis values. This property can be set to 'auto'. For host or file scopes, this parameter has no effect.	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   get | set | xpctarget.xpcsc.get (scope object) |
xpctarget.xpc.set (target application object) |
xpctarget.xpc.get (target application object)

xpctarget.xpcsc.start (scope object)

Purpose   Start execution of scope on target computer

Syntax   **MATLAB command line**

```
start(scope_object_vector)
scope_object_vector.start
+scope_object_vector
start(getscope((target_object, signal_index_vector))
```

Target computer command line

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

Arguments

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

Description   Method for a scope object. Starts a scope on the target computer represented by a scope object on the host computer. This method might not start data acquisition, which depends on the trigger settings. Before using this method, you must create a scope. To create a scope, use the target object method addscope or add xPC Target scope blocks to your Simulink model.

Examples   Start one scope with the scope object sc1.

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

or type

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

Start two scopes.

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

or type

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

or type

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

Start all scopes:

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

or type

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

See Also   xpctarget.xpc.getscope | xpctarget.xpc.stop (target
application object) | xpctarget.xpcsc.stop (scope object)

xpctarget.xpcsc.stop (scope object)

Purpose   Stop execution of scope on target computer

Syntax   **MATLAB command line**

stop(scope_object_vector)
scope_object.stop
-scope_object
stop(getscope(target_object, signal_index_vector))

Target computer command line

stopscope scope_index
stopscope 'all'

Arguments

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

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

Examples   Stop one scope represented by the scope object sc1.

stop(sc1) or sc1.stop or -sc1

Stop all scopes with a scope object vector allscopes created with the command

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

or type

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

See Also   xpctarget.xpc.getscope | xpctarget.xpc.stop (target
application object) | xpctarget.xpc.start (target application
object) | xpctarget.xpcsc.start (scope object)

xpctarget.xpcsc.trigger

Purpose	Software-trigger start of data acquisition for scope(s)
Syntax	`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)

xpctarget.xpcschost Class

Purpose Control and access properties of host scopes

Description The scope gets a data package from the kernel, waits for an upload command from the host computer, and uploads the data to the host. The host computer displays the data using a scope viewer or other MATLAB functions.

Methods

These methods are inherited from xpctarget.xpcsc Class.

Method	Description
xpctarget.xpcsc.addsignal	Add signals to scope represented by scope object
xpctarget.xpcsc.get (scope object)	Return property values for scope objects
xpctarget.xpcsc.remsignal	Remove signals from scope represented by scope object
xpctarget.xpcsc.set (scope object)	Change property values for scope objects
xpctarget.xpcsc.start (scope object)	Start execution of scope on target computer
xpctarget.xpcsc.stop (scope object)	Stop execution of scope on target computer
xpctarget.xpcsc.trigger	Software-trigger start of data acquisition for scope(s)

Properties

These properties are inherited from xpctarget.xpcsc Class.

Property	Description	Writable
Application	Name of the Simulink model associated with this scope object.	No
Decimation	A number n, where every nth sample is acquired in a scope window.	Yes

Property	Description	Writable
NumPrePostSamples	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.	Yes
NumSamples	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. For file scopes, this parameter works in conjunction with the **AutoRestart** check box. If the **AutoRestart** box is selected, the file scope collects data up to **Number of Samples**, then starts over again, overwriting the buffer. If the **AutoRestart** box is not selected, the file scope collects data only up to **Number of Samples**, then stops.	Yes
ScopeId	A numeric index, unique for each scope.	No
Signals	List of signal indices from the target object to display on the scope.	Yes
Status	Indicate whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are 'Acquiring', 'Ready for being Triggered', 'Interrupted', and 'Finished'.	No

xpctarget.xpcschost Class

Property	Description	Writable
TriggerLevel	If TriggerMode is 'Signal', indicates the value the signal has to cross to trigger the scope and start acquiring data. The trigger level can be crossed with either a rising or falling signal.	Yes
TriggerMode	Trigger mode for a scope. Valid values are 'FreeRun' (default), 'Software', 'Signal', and 'Scope'.	Yes
TriggerSample	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. 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.	Yes
TriggerScope	If TriggerMode is 'Scope', identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered. You do this by setting the slave scope property TriggerScope to the scope index of the master scope.	Yes

Property	Description	Writable
TriggerSignal	If `TriggerMode` is `'Signal'`, identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property `Signal`.	Yes
TriggerSlope	If `TriggerMode` is `'Signal'`, indicates whether the trigger is on a rising or falling signal. Values are `'Either'` (default), `'Rising'`, and `'Falling'`.	Yes
Type	Determines whether the scope is displayed on the host computer or on the target computer. Values are `'Host'`, `'Target'`, and `'File'`.	Yes

These properties are specific to class `xpcschost`.

Property	Description	Writeable
Data	Contains the output data for a single data package from a scope. For target or file scopes, this parameter has no effect.	No
Time	Contains the time data for a single data package from a scope. For target or file scopes, this parameter has no effect.	No

xpctarget.xpcsctg Class

Purpose   Control and access properties of target scopes

Description   The kernel acquires a data package and the scope displays the data on the target computer screen. Depending on the setting of DisplayMode, the data may be displayed numerically or graphically by a redrawing, sliding, and rolling display.

Methods

These methods are inherited from xpctarget.xpcsc Class.

Method	Description
xpctarget.xpcsc.addsignal	Add signals to scope represented by scope object
xpctarget.xpcsc.get (scope object)	Return property values for scope objects
xpctarget.xpcsc.remsignal	Remove signals from scope represented by scope object
xpctarget.xpcsc.set (scope object)	Change property values for scope objects
xpctarget.xpcsc.start (scope object)	Start execution of scope on target computer
xpctarget.xpcsc.stop (scope object)	Stop execution of scope on target computer
xpctarget.xpcsc.trigger	Software-trigger start of data acquisition for scope(s)

Properties

These properties are inherited from xpctarget.xpcsc Class.

Property	Description	Writable
Application	Name of the Simulink model associated with this scope object.	No
Decimation	A number n, where every nth sample is acquired in a scope window.	Yes

Property	Description	Writable
NumPrePostSamples	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.	Yes
NumSamples	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. For file scopes, this parameter works in conjunction with the **AutoRestart** check box. If the **AutoRestart** box is selected, the file scope collects data up to **Number of Samples**, then starts over again, overwriting the buffer. If the **AutoRestart** box is not selected, the file scope collects data only up to **Number of Samples**, then stops.	Yes
ScopeId	A numeric index, unique for each scope.	No
Signals	List of signal indices from the target object to display on the scope.	Yes
Status	Indicate whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are 'Acquiring', 'Ready for being Triggered', 'Interrupted', and 'Finished'.	No

Property	Description	Writable
TriggerLevel	If TriggerMode is 'Signal', indicates the value the signal has to cross to trigger the scope and start acquiring data. The trigger level can be crossed with either a rising or falling signal.	Yes
TriggerMode	Trigger mode for a scope. Valid values are 'FreeRun' (default), 'Software', 'Signal', and 'Scope'.	Yes
TriggerSample	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. 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.	Yes
TriggerScope	If TriggerMode is 'Scope', identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered. You do this by setting the slave scope property TriggerScope to the scope index of the master scope.	Yes

Property	Description	Writable
TriggerSignal	If TriggerMode is 'Signal', identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property Signal.	Yes
TriggerSlope	If TriggerMode is 'Signal', indicates whether the trigger is on a rising or falling signal. Values are 'Either' (default), 'Rising', and 'Falling'.	Yes
Type	Determines whether the scope is displayed on the host computer or on the target computer. Values are 'Host', 'Target', and 'File'.	Yes

These properties are specific to class xpcsctg.

Property	Description	Writeable
DisplayMode	For target scopes, indicate how a scope displays the signals. Values are 'Numerical', 'Redraw' (default), 'Sliding', and 'Rolling'. For host or file scopes, this parameter has no effect. .	Yes
Grid	Values are 'on' and 'off'. For host or file scopes, this parameter has no effect.	Yes

xpctarget.xpcsctg Class

Property	Description	Writeable
Mode	**Note** The Mode property will be removed in a future release. • For target scopes, use DisplayMode. • For file scopes, use WriteMode. • For host scopes, this parameter has no effect.	Yes
YLimit	Minimum and maximum *y*-axis values. This property can be set to 'auto'. For host or file scopes, this parameter has no effect.	Yes

Purpose   Test communication between host and target computers

Syntax   xpctargetping
               xpctargetping('mode', 'arg1', 'arg2')

Arguments

mode	Optionally, enter the communication mode:

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

	TCPIP	Enable TCP/IP connection with target computer.
	RS232	Enable RS-232 connection with target computer.
arg1	Optionally, enter an argument based on the mode value:	
	IP address	If mode is 'TCPIP', enter the IP address of the target computer.
	COM port	If mode is 'RS232', enter the host COM port.
arg2	Optionally, enter an argument based on the mode value:	
	Port	If mode is 'TCPIP', enter the port number for the target computer.
	Baud rate	If mode is 'RS232', enter the baud rate for the connection between the host and target computer.

xpctargetping

Description   Pings the target computer from the host computer and returns either success or failed. If you have one target computer, or if you designate a target computer as the default one in your system, use the syntax

xpctargetping

If you have multiple target computers in your system, use the following syntax to identify the target computer to ping.

xpctargetping('mode', 'arg1', 'arg2')

This function returns success if the xPC Target kernel is loaded and running and communication is working between host and target.

This function works with both RS-232 and TCP/IP communication.

Examples   Check for communication between the host computer and target computer.

xpctargetping

If you have a serial connection with the target computer you want to check, use the following syntax.

xpctargetping('RS232', 'COM1', '115200')

How To   • "Run Confidence Test on Configuration"

Purpose   Open Real-Time xPC Target Spy window on host computer

Syntax   xpctargetspy
                  xpctargetspy(target_object)
                  xpctargetspy('target_object_name')

Arguments   target_object   Variable name to reference the target object.

                  target_object_name   Target object name as specified in the xPC
                                       Target Explorer.

Description   This graphical user interface (GUI) allows you to upload displayed data
                  from the target computer. By default, xpctargetspy opens a Real-Time
                  xPC Target Spy window for the target object, tg. If you have multiple
                  target computers in your system, you can call the xpctargetspy
                  function for a particular target object, target_object.

                  If you have one target computer, or if you designate a target computer
                  as the default one in your system, use the syntax

                  xpctargetspy

                  If you have multiple target computers in your system, use
                  xpctarget.xpc to create the additional target object first.

                  target_object=xpctarget.xpc('mode', 'arg1', 'arg2')

                  Then, use the following syntax.

                  xpctargetspy(target_object)

                  If you have a target computer object in the xPC Target Explorer, you
                  can use the following syntax.

                  target_object=xpctarget.xpc('target_object_name')

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

- If `TargetScope` is enabled, a single graphics screen is uploaded. The screen is not continually updated because of a higher data volume when a target graphics card is in VGA mode. You must explicitly request an update. To manually update the host screen with another target screen, move the pointer into the Real-Time xPC Target Spy window and right-click to select **Update xPC Target Spy**.

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

Examples   To open the Real-Time xPC Target Spy window for a default target computer, `tg`, in the MATLAB window, type

```
xpctargetspy
```

To open the Real-Time xPC Target Spy window for a target computer, tg1, in the MATLAB window, type

```
xpctargetspy(tg1)
```

If you have multiple target computers in your system, use `xpctarget.xpc` to create the additional target object, tg2, first.

```
tg2=xpctarget.xpc('RS232', 'COM1', '115200')
```

Then, use the following syntax.

```
xpctargetspy(tg2)
```

Purpose   Test xPC Target installation

Syntax   xpctest
                xpctest('noreboot')
                xpctest('-noreboot')
                xpctest('target_name')
                xpctest('target_name','noreboot')
                xpctest('target_name','-noreboot')

Arguments   'target_name'   Name of target computer to test.

                'noreboot'   Only one possible option. Skips the reboot test.
                                Use this option if the target hardware does not
                                support software rebooting. Value is 'noreboot'
                                or '-noreboot'.

Description xpctest is a series of xPC Target tests to check the functioning of the
                following xPC Target tasks:

- Initiate communication between the host and target computers.

- Reboot the target computer to reset the target environment.

- Build a target application on the host computer.

- Download a target application to the target computer.

- Check communication between the host and target computers using commands.

- Execute a target application.

- Compare the results of a simulation and the target application run.

xpctest('noreboot') or xpctest('-noreboot') skips the reboot test
on the default target computer. Use this option if target hardware does
not support software rebooting.

xpctest('target_name') runs the tests on the target computer
identified by 'target_name'.

xpctest

xpctest('target_name','noreboot') or
xpctest('target_name','-noreboot') runs the tests on the target
computer identified by 'target_name', but skips the reboot test.

Examples   If the target hardware does not support software rebooting, or to skip
the reboot test, in the MATLAB window, type

xpctest('-noreboot')

To run xpctest on a specified target computer, for example TargetPC1,
type

xpctest('TargetPC1')

How To   • "Run Confidence Test on Configuration"

• "Test 1: Ping Using System Ping" on page 13-2

Purpose   Disconnect target computer from current client application

Syntax   xpcwwwenable
                   xpcwwwenable('target_obj_name')

Description   xpcwwwenable disconnects the target application from the MATLAB
                   interface so you can connect to the Web browser. You can also use this
                   function to connect to the MATLAB interface after using a Web browser,
                   or to switch to another Web browser.

                   xpcwwwenable('target_obj_name') disconnects the target application
                   on target_obj_name (for example 'TargetPC1')from the MATLAB
                   interface.

xpcwwwenable

28

Configuration Parameters

This topic deals with configuration parameters in xPC Target Explorer and in the MATLAB API.

Setting Configuration Parameters

xPC Target options Pane

Set up general information about building target applications, including target, execution, data logging, and other options.

Configuration

To enable the xPC Target options pane, you must:

1 Select `xpctarget.tlc` or `xpctargetert.tlc` for the **System target file** parameter on the code generation pane.

2 Select `C` for the **Language** parameter on the code generation pane.

Tips

- The default xPC Target options work for the generation of most target applications. If you want to customize the build of your target application, set the option parameters to suit your specifications.

- To access these parameters from the MATLAB command line, use:
  - `gcs` — To access the current model.
  - `set_param` — To set the parameter value.
  - `get_param` — To get the current value of the parameter.

See Also

"xPC Target Options Configuration Parameters" on page 4-3

Automatically download application after building

Enable Simulink Coder to build and download the target application to the target computer.

Settings

Default: on

☑ On

Builds and downloads the target application to the target computer.

☐ Off

Builds the target application, but does not download it to the target computer.

Command-Line Information

Parameter: xPCisDownloadable
Type: string
Value: 'on' | 'off'
Default: 'on'

See Also

"Build and Download Target Application"

Download to default target PC

Direct Simulink Coder to download the target application to the default target computer.

Settings

Default: on

☑ On

Downloads the target application to the default target computer. Assumes that you configured a default target computer through xPC Target Explorer.

☐ Off

Enables the **Specify target PC name** field so that you can enter the target computer to which to download the target application.

Dependency

This parameter enables **Specify target PC name**.

Command-Line Information

Parameter: xPCisDefaultEnv
Type: string
Value: 'on' | 'off'
Default: 'on'

See Also

• "Ethernet Communication Setup"

• "RS-232 Communication Setup"

Specify target PC name

Specify a target computer name for your target application.

Settings

' '

Tip

The target computer name appears in xPC Target Explorer as the target computer node, for example `TargetPC1`.

Dependencies

This parameter is enabled by **Download to default target PC**.

Command-Line Information

Parameter: `xPCTargetPCEnvName`
Type: string
Value: Any valid target computer
Default: ' '

See Also

"xPC Target Explorer" on page 4-4

Name of xPC Target object created by build process

Enter the name of the target object created by the build process.

Settings

Default: `tg`

Tip

Use this name when you work with the target object through the command-line interface.

Command-Line Information

Parameter: `RL32ObjectName`
Type: string
Value: `'tg'` | valid target object name
Default: `'tg'`

See Also

"Target Driver Objects" on page 7-2

Use default communication timeout

Direct xPC Target software to wait 5 (default) seconds for the target application to be downloaded to the target computer.

Settings

Default: on

☑ On

> Waits the default amount of seconds (5) for the target application to be downloaded to the target computer.

☐ Off

> Enables the **Specify the communication timeout in seconds** field so that you can enter the maximum length of time in seconds you want to wait for a target application to be downloaded to the target computer.

Dependencies

This parameter enables **Specify the communication timeout in seconds**.

Command-Line Information

> **Parameter:** xPCisModelTimeout
> **Type:** string
> **Value:** 'on' | 'off'
> **Default:** 'on'

See Also

"Increase the Time for Downloads" on page 20-4

Specify the communication timeout in seconds

Specify a timeout, in seconds, to wait for the target application to download to the target computer.

Settings

Default: 5

Tip

Enter the maximum length of time in seconds you want to allow the xPC Target software to wait for the target application to download to the target computer. If the target application is not downloaded within this time frame, the software generates an error.

Dependencies

This parameter is enabled by **Use default communication timeout**.

Command-Line Information

> **Parameter:** xPCModelTimeoutSecs
> **Type:** string
> **Value:** Any valid number of seconds
> **Default:** '5'

See Also

"Increase the Time for Downloads" on page 20-4

Execution mode

Specify target application execution mode.

Settings

Default: `Real-Time`

`Real-Time`
> Executes target application in real time.

`Freerun`
> Runs the target application as fast as possible.

Command-Line Information

   Parameter: `RL32ModeModifier`
   Type: string
   Value: `'Real-Time'` | `'Freerun'`
   Default: `'Real-Time'`

See Also

"Set Configuration Parameters"

Real-time interrupt source

Select a real-time interrupt source from the I/O board.

Settings

Default: Timer

Timer
> Specifies that the board interrupt source is a timer.

Auto (PCI only)
> Enables the xPC Target software to automatically determine the IRQ that the BIOS assigned to the board and use it.

3 to 15
> Specifies that the board interrupt source is an IRQ number on the board.

Tips

- The Auto (PCI only) option is available only for PCI boards. If you have an ISA board (PC 104 or onboard parallel port), you must set the IRQ manually.

- The xPC Target software treats PCI parallel port plug-in boards like ISA boards. For PCI parallel port plug-in boards, you must set the IRQ manually.

- Multiple boards can share the same interrupt number.

Command-Line Information

Parameter: RL32IRQSourceModifier
Type: string
Value: 'Timer' | Auto (PCI only) | '3'|'4'|'5' | '6'|'7' |'8' |'9' |'10' |'11' |'12' |'13' |'14' |'15'
Default: 'Timer'

See Also

"Set Configuration Parameters"

I/O board generating the interrupt

Specify the board interrupt source.

Settings

Default: None/Other

ATI-RP-R5
> Specifies that the interrupt source is an ATI-RP-R5 board.

AudioPMC+
> Specifies that the interrupt source is the Bittware AudioPMC+ audio board.

Bitflow NEON
> Specifies that the interrupt source is the BitFlow™ NEON video board.

CB_CIO-CTR05
> Specifies that the interrupt source is the Measurement Computing™ CIO-CTR05 board.

CB_PCI-CTR05
> Specifies that the interrupt source is the Measurement Computing PCI-CTR05 board.

Diamond_MM-32
> Specifies that the interrupt source is the Diamond Systems MM-32 board.

FastComm 422/2-PCI
> Specifies that the interrupt source is the Fastcom 422/2-PCI board.

FastComm 422/2-PCI-335
> Specifies that the interrupt source is the Fastcom 422/2-PCI-335 board.

FastComm 422/4-PCI-335
> Specifies that the interrupt source is the Fastcom 422/4-PCI-335 board.

GE_Fanuc(VMIC)_PCI-5565
> Specifies that the interrupt source is the GE® Fanuc VMIC PCI-5565 board.

General Standards 24DSI12
> Specifies that the interrupt source is the General Standards 24DSI12 board.

Parallel_Port
> Specifies that the interrupt source is the parallel port of the target computer.

Quatech DSCP-200/300
> Specifies that the interrupt source is the Quatech® DSCP-200/300 board.

Quatech ESC-100
> Specifies that the interrupt source is the Quatech ESC-100 board.

Quatech QSC-100
> Specifies that the interrupt source is the Quatech QSC-100 board.

Quatech QSC-200/300
> Specifies that the interrupt source is the Quatech QSC-200/300 board.

RTD_DM6804
> Specifies that the interrupt source is the Real-Time Devices DM6804 board.

SBS_25x0_ID_0x100
> Specifies that the interrupt source is an SBS Technologies shared memory board associated with ID 0x100.

SBS_25x0_ID_0x101
> Specifies that the interrupt source is an SBS Technologies shared memory board associated with ID 0x101.

SBS_25x0_ID_0x102
> Specifies that the interrupt source is an SBS Technologies shared memory board associated with ID 0x102.

SBS_25x0_ID_0x103
> Specifies that the interrupt source is an SBS Technologies shared memory board associated with ID 0x103.

Scramnet_SC150+
> Specifies that the interrupt source is the Systran® Scramnet+ SC150 board.

Softing_CAN-AC2-104
> Specifies that the interrupt source is the Softing® CAN-AC2-104 board.

Softing_CAN-AC2-PCI
> Specifies that the interrupt source is the Softing CAN-AC2-PCI board.

28-13

Speedgoat_IO301
     Specifies that the interrupt source is the Speedgoat IO301 FPGA board.

Speedgoat_IO302
     Specifies that the interrupt source is the Speedgoat IO302 FPGA board.

Speedgoat_IO303
     Specifies that the interrupt source is the Speedgoat IO303 FPGA board.

Speedgoat_IO311
     Specifies that the interrupt source is the Speedgoat IO311 FPGA board.

Speedgoat_IO312
     Specifies that the interrupt source is the Speedgoat IO312 FPGA board.

Speedgoat_IO313
     Specifies that the interrupt source is the Speedgoat IO313 FPGA board.

Speedgoat_IO314
     Specifies that the interrupt source is the Speedgoat IO314 FPGA board.

Speedgoat_IO325
     Specifies that the interrupt source is the Speedgoat IO325 FPGA board.

Speedgoat_IO331
     Specifies that the interrupt source is the Speedgoat IO331 FPGA board.

UEI_MFx
     Specifies that the interrupt source is a United Electronic Industries
     UEI-MF series board.

None/Other
     Specifies that the I/O board has no interrupt source.

Command-Line Information

Parameter: xPCIRQSourceBoard
Type: string
Value: 'ATI-RP-R5' |
'AudioPMC+' |
'Bitflow NEON' |
'CB_CIO-CTR05' |
'CB_PCI-CTR05' |
'Diamond_MM-32' |
'FastComm 422/2-PCI' |

```
'FastComm 422/2-PCI-335' |
'FastComm 422/4-PCI-335' |
'GE_Fanuc(VMIC)_PCI-5565' |
'General Standards 24DSI12' |
'Parallel_Port' |
'Quatech DSCP-200/300' |
'Quatech ESC-100' |
'Quatech QSC-100' |
'Quatech QSC-200/300' |
'RTD_DM6804' |
'SBS_25x0_ID_0x100' |
'SBS_25x0_ID_0x101' |
'SBS_25x0_ID_0x102' |
'SBS_25x0_ID_0x103' |
'Scramnet_SC150+' |
'Softing_CAN-AC2-104' |
'Softing_CAN-AC2-PCI' |
'Speedgoat_IO301' |
'Speedgoat_IO302' |
'Speedgoat_IO303' |
'Speedgoat_IO311' |
'Speedgoat_IO312' |
'Speedgoat_IO313' |
'Speedgoat_IO314' |
'Speedgoat_IO325' |
'Speedgoat_IO331' |
'UEI_MFx' |
'None/Other'
```
Default: `'None/Other'`

See Also
"Set Configuration Parameters"

PCI slot (-1: autosearch) or ISA base address

Enter the slot number or base address for the I/O board generating the interrupt.

Settings
Default: -1

The PCI slot can be either -1 (let the xPC Target software determine the slot number) or of the form [bus, slot].

The base address is a hexadecimal number of the form 0x300.

Tip
To determine the bus and PCI slot number of the boards in the target computer, type getxpcpci in the MATLAB window.

Command-Line Information

Parameter: xPCIOIRQSlot
Type: string
Value: '-1' | hexadecimal value
Default: '-1'

See Also
"xPC Target Options Configuration Parameters" on page 4-3

"PCI Bus I/O Devices"

Log Task Execution Time

Log task execution times to the target object property `tg.TETlog`.

Settings

Default: on

☑ On

Logs task execution times to the target object property `tg.TETlog`.

☐ Off

Does not log task execution times to the target object property `tg.TETlog`.

Command-Line Information

Parameter: `RL32LogTETModifier`
Type: string
Value: `'on'` | `'off'`
Default: `'on'`

See Also

"xPC Target Options Configuration Parameters" on page 4-3

"Signal Logging Basics" on page 5-77

Signal logging data buffer size in doubles

Enter the maximum number of sample points to save before wrapping.

Settings

Default: 100000

The maximum value for this option cannot exceed the available target computer memory, which the xPC Target software also uses to hold other items.

Tips

- Target applications use this buffer to store the time, states, outputs, and task execution time logs as defined in the Simulink model.

- The maximum value for this option derives from available target computer memory, which the xPC Target software also uses to hold other items. For example, in addition to signal logging data, the software also uses the target computer memory for the xPC Target kernel, target application, and scopes.

  For example, assume that your model has six data items (one time, two states, two outputs, and one task execution time (TET)). If you enter a buffer size of `100000`, the target object property `tg.MaxLogSamples` is calculated as `floor(100000) / 6) = 16666`. After saving 16666 sample points, the buffer wraps and further samples overwrite the older ones.

- If you enter a logging buffer size larger than the available RAM on the target computer, after downloading and initializing the target application, the target computer displays a message, `ERROR: allocation of logging memory failed`. To avoid this error, either install more RAM or reduce the buffer size for logging, and then reboot the target computer. To calculate the maximum buffer size you might have for your target application logs, divide the amount of available RAM on your target computer by `sizeof(double)`, or `8`. Enter that value for the **Signal logging data buffer size in doubles** value.

Command-Line Information

Parameter: RL32LogBufSizeModifier
Type: string

Value: '100000' | any valid memory size
Default: '100000'

See Also

"xPC Target Options Configuration Parameters" on page 4-3

Enable profiling

Enable profiling and visual presentation of target computer execution.

Settings

Default: off

☑ On

Profile target computer execution.

☐ Off

Do not profile target computer execution.

Tips

- Before building and downloading a model, select this check box to observe the target computer thread execution.

- If you are using multiple CPU cores on a target computer, select **Enabling profiling** to verify that the xPC Target software is actually executing on the multiple CPU cores.

Tip For more on configuring your model for concurrent execution, see "Design Considerations".

Command-Line Information

Parameter: xPCTaskExecutionProfile
Type: string
Value: 'on' | 'off'
Default: 'off'

See Also

"Profiling Target Application Execution" on page 26-6

Number of events (each uses 20 bytes)

Enter the maximum of events to log for the profiling tool.

Settings

Default: 5000

The maximum number of events to be logged for the profiling tool.

Tips

- An event is the start of end of an interrupt or iteration of the model. For example, one sample can four events: the beginning and end of an interrupt, and the beginning and end of an iteration.

- Use this parameter in conjunction with the **Enable profiling** parameter.

- Each event contains information such as the CPU ID, model thread ID (TID), event ID, and time stamp readings. Each event occupies 20 bytes.

Command-Line Information

Parameter: xPCRL32EventNumber
Type: string
Value: any valid number of events
Default: '5000'

See Also

"Profiling Target Application Execution" on page 26-6

Double buffer parameter changes

Use a double buffer for parameter tuning. This enables parameter tuning so that the process of changing parameters in the target application uses a double buffer.

Settings

Default: off

☑ On

Changes parameter tuning to use a double buffer.

☐ Off

Suppresses double buffering of parameter changes in the target application.

Tips

- When a parameter change request is received, the new value is compared to the old one. If the new value is identical to the old one, it is discarded, and if different, it is queued.

- At the start of execution of the next sample of the real-time task, the queued parameters are updated. This means that parameter tuning affects the task execution time (TET), and the very act of parameter tuning can cause a CPU overload error.

- Double buffering leads to a more robust parameter tuning interface, but it increases Task Execution Time (TET) and the higher probability of overloads. Under typical conditions, keep double buffering off (default).

Command-Line Information

Parameter: `xpcDblBuff`
Type: string
Value: `'on'` | `'off'`
Default: `'off'`

See Also

"xPC Target Options Configuration Parameters" on page 4-3

Load a parameter set from a file on the designated target file system

Automatically load a parameter set from a file on the designated target computer file system.

Settings

Default: off

☑ On

Enable the automatic loading of a parameter set from the file specified by **File name** on the designated target computer file system.

☐ Off

Suppress the automatic loading of a parameter set from a file on the designated target computer file system.

Dependencies

This parameter enables **File name**.

Command-Line Information

Parameter: xPCLoadParamSetFile
Type: string
Value: 'on' | 'off'
Default: 'off'

See Also

"xPC Target Options Configuration Parameters" on page 4-3

"Save and Reload Parameters Using MATLAB Language" on page 5-128

File name

Specify the target computer file name from which to load the parameter set.

Settings

' '

Tip

If the named file does not exist, the software loads the parameter set built with the model.

Dependencies

This parameter is enabled by **Load a parameter set from a file on the designated target file system**.

Command-Line Information

Parameter: xPCOnTgtParamSetFileName
Type: string
Value: Any valid file name
Default: ' '

See Also

"xPC Target Options Configuration Parameters" on page 4-3

Build COM objects from tagged signals/parameters

Enable build process to create a model-specific COM library file.

Settings

Default: off

☑ On

Creates a model-specific COM library file, `<model_name>COMiface.dll`.

☐ Off

Does not create a model-specific COM library file.

Tip

Use the model-specific COM library file to create custom GUIs with Visual Basic® or other tools that can use COM objects.

Command-Line Information

Parameter: `xpcObjCom`
Type: string
Value: `'on'` | `'off'`
Default: `'off'`

See Also

"Creating the Target Application and Model-Specific COM Library"

Generate CANape extensions

Enable target applications to generate data, such as that for A2L, for Vector CANape.

Settings

Default: off

☑ On

Enables target applications to generate data, such as that for A2L, for Vector CANape.

☐ Off

Does not enable target applications to generate data, such as that for A2L, for Vector CANape.

Command-Line Information

Parameter: xPCGenerateASAP2
Type: string
Value: 'on' | 'off'
Default: 'off'

See Also

"Configuring the Vector CANape Device" on page 2-6

Include model hierarchy on the target application

Includes the Simulink model hierarchy as part of the target application.

Settings

Default: off

☑ On

Includes the model hierarchy as part of the target application.

☐ Off

Excludes the model hierarchy from the target application.

Tips

Including the model hierarchy in the target application:

- Lets you connect to the target computer from xPC Target Explorer without being in the target application build folder.

- Can increase the size of the target application, depending on the size of the model.

Command-Line Information

Parameter: `xPCGenerateXML`
Type: string
Value: `'on'` | `'off'`
Default: `'off'`

See Also

"Monitor Signals Using xPC Target Explorer" on page 5-5

Enable Stateflow animation

Enables visualization of Stateflow chart animation.

Settings

Default: off

☑ On

Enables visualization of Stateflow chart animation.

☐ Off

Disables visualization of Stateflow chart animation.

Command-Line Information

Parameter: xPCEnableSFAnimation
Type: string
Value: 'on' | 'off'
Default: 'off'

See Also

"Animate Stateflow Charts Using Simulink External Mode" on page 5-16