xPC Target

For Use with Real-Time Workshop®

- Modeling
- Simulation
- Implementation

User's Guide

Version 2

The MathWorks

How to Contact The MathWorks:

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup
`support@mathworks.com`	Technical support
`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information
508-647-7000	Phone
508-647-7001	Fax
The MathWorks, Inc. 3 Apple Hill Drive Natick, MA 01760-2098	Mail

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

xPC Target User's Guide

Printing History:	September 1999	First printing	New for Version 1 (Release 11.1)
	November 2000	Online only	Revised for Version 1.1 (Release 12)
	June 2001	Online only	Revised for Version 1.2 (Release 12.1)
	September 2001	Online only	Revised for Version 1.3 (Release 12.1+)
	July 2002	Online only	Revised for Version 2 (Release 13)
	June 2004	Online only	Revised for Version 2.5 (Release 14)
	August 2004	Online only	Revised for Version 2.6 (Release 14+)
	October 2004	Online only	Revised for Version 2.6.1 (Release 14SP1)
	November 2004	Online only	Revised for Version 2.7 (Release 14SP1+)
	March 2005	Online only	Revised for Version 2.7.2 (Release 14SP2)

Contents

1 Target and Scope Objects

2 Targets and Scopes in the MATLAB Interface

3 Signals and Parameters

4 Embedded Option

5 Software Environment and Demos

6 Using the Target PC Command-Line Interface

7 Working with Target PC Files and File Systems

8 Graphical User Interfaces

9 xPC Target Web Browser Interface

10 Interrupts Versus Polling

11 xPC Target and Fortran

12 Troubleshooting

13 Target PC Command-Line Interface Reference

14 Function Reference

Index

1

Target and Scope Objects

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

Target Objects

xPC Target uses a target object (of class `xpctarget.xpc`) to represent the target kernel and your target application. Use target object functions to run and control real-time applications on the target PC with scope objects to collect signal data.

See Chapter 14, “Function Reference,” for a reference of the target functions.

What Is a Target Object?

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

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

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

To create a target object,

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

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

Scope Objects

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

See Chapter 14, “Function Reference,” for a reference of the scope functions.

What Is a Scope Object?

xPC Target uses scopes and scope objects as an alternative to using Simulink® scopes and external mode. A scope can exist as part of a Simulink model system or outside a model system.

- A scope that is part of a Simulink model system is a scope block. You add an xPC Target scope block to the model, build an application from that model, and download that application to the target PC.
- A scope that is outside a model is not a scope block. For example, if you create a scope with the `addscope` method, that scope is not part of a model system. You add this scope to the model after the model has been downloaded and initialized.

This difference affects when and how the scope executes to acquire data.

Scope blocks inherit sample times. A scope block in the root model or a normal subsystem executes at the sample time of its input signals. A scope block in a conditionally executed (triggered/enabled) subsystem executes whenever the containing subsystem executes. Note that in the latter case, the scope might acquire samples at irregular intervals.

A scope that is not part of a model always executes at the base sample time of the model. Thus, it might acquire repeated samples. For example, if the model base sample time is 0.001, and you add to the scope a signal whose sample time is 0.005, the scope will acquire five identical samples for this signal, and then the next five identical samples, and so on.

Understanding the structure of scope objects will help you to use the MATLAB command-line interface to view and collect signal data. The topics in this section are

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

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

To create a scope object,

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

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

The following section describes scope object types.

Scope Object Types

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

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

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

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

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

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

- Data
- StartTime
- Time

The xpcfs scope object of type file has the following object properties:

- AutoRestart
- Filename
- Mode
- StartTime
- WriteSize

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

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

See the scope object function get (scope object) on page 14-28 for a description of these object properties.

2

Targets and Scopes in the MATLAB Interface

You can work with xPC Target target and scope objects through the MATLAB interface (MATLAB Command Window), the target PC command line, a Web browser, or an xPC Target API. This chapter describes how to use the MATLAB interface to work with the target and scope objects in the following topics.

See Chapter 6, “Using the Target PC Command-Line Interface,” for a description of the target PC command-line interface.

Working with Target Objects

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

See Chapter 14, "Function Reference," for a reference of the target object functions.

Creating Target Objects

To create a target object,

- Build a target application. xPC Target creates a target object during the build process.
- To create a single target object, or to create multiple target objects in your system, use the target object constructor function `xpc` (see `xpctarget.xpc` on page 14-116) as follows. For example, the following creates a target object connected to the host through an RS-232 connection. In the MATLAB window, type

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

  The resulting target object is `tg`.

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

```
tg.targetping
```

Note To ensure that you always know which target PC is associated with your target object, you should always use this method to create target objects.

- To create a single target object, or to create the first of many targets in your system, use the target object constructor function `xpctarget.xpc` as follows. In the MATLAB Command Window, type

  ```
  tg = xpctarget.xpc
  ```

  The resulting target object is `tg`.

Note If you choose to use this syntax to create a target object, you should use **xPC Target Explorer** to configure your target PC. This ensures that command-line interactions know the correct target PC to work with.

Deleting Target Objects

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

```
tg.delete
```

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

Displaying Target Object Properties

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

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

1 In the MATLAB window, type

```
tg
```

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

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

2 Type

```
+tg
```

The `Status` property changes from `stopped` to `running`, and the log properties change to `Acquiring`.

For a list of target object properties with a description, see the target object function `get (target object)` on page 14-37

Setting Target Object Properties from the Host PC

You can change a target object property by using the xPC Target `set` method or the dot notation on the host PC.

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

```
target_object.property_name = new_property_value
```

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

1 In the MATLAB window, type

```
tg.stoptime = 1000
```

2 Alternatively, you can type

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

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

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

Getting the Value of a Target Object Property

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

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

```
target_object.property_name
```

For example, to access the start time,

1 In the MATLAB window, type

```
endrun = tg.stoptime
```

2 Alternatively, you can type

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

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

Signals are not target object properties. To get the value of the `Integrator1` signal from the model `xpcosc`,

1 In the MATLAB window, type

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

where `0` is the signal index.

2 Alternatively, you could type

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

Note Method names are case sensitive. You must type the entire name. Property names are not case sensitive. You do not need to type the entire name as long as the characters you do type are unique for the property.

Using the Method Syntax with Target Objects

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

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

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

1. In the MATLAB window, type

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

2. Alternatively, you can type

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

Working with Scope Objects

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

- "Displaying Scope Object Properties for a Single Scope" on page 2-7
- "Displaying Scope Object Properties for All Scopes" on page 2-8
- "Setting the Value of a Scope Property" on page 2-8
- "Getting the Value of a Scope Property" on page 2-9
- "Using the Method Syntax with Scope Objects" on page 2-10
- "Acquiring Signal Data with Scopes of Type File" on page 2-11
- "Acquiring Gap-Free Data Using Two Scopes" on page 2-16
- "Acquiring Gap-Free Data Using Two Scopes" on page 2-16

See Chapter 14, "Function Reference," for a reference of the scope object functions.

Displaying Scope Object Properties for a Single Scope

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

1 In the MATLAB window, type

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

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

2 Type

```
sc1
```

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

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

For a list of target object properties with a description, see the target function `get (target object)` on page 14-37.

Displaying Scope Object Properties for All Scopes

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

1 In the MATLAB window, type

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

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

2 Alternatively, type

   ```
   allscopes = getscope(tg)
   ```

   or type

   ```
   allscopes = tg.getscope
   ```

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

For a list of target object properties with a description, see the target function `get (target object)` on page 14-37

Setting the Value of a Scope Property

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

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

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

1 In the MATLAB window, type

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

2 Alternatively, you can type

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

or type

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

Note that you cannot use dot notation to set vector object properties. To assign properties to a vector of scopes, use the `set` method. For example, assuming you have a variable `sc12` for two scopes, `1` and `2`, set the `NumSamples` property of these scopes to `300`:

1 In the MATLAB window, type

```
set(sc12,'NumSamples',300)
```

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

Note Method names are case sensitive. You must type the entire name. Property names are not case sensitive. You do not need to type the entire name as long as the characters you do type are unique for the property.

Getting the Value of a Scope Property

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

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

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

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

1 In the MATLAB window, type

```
beginrun = sc1.starttime
```

2 Alternatively, you can type

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

or type

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

Note that you cannot use dot notation to get the values of vector object properties. To get properties of a vector of scopes, use the `get` method. For example, assume you have two scopes, `1` and `2`, assigned to the variable `sc12`.

To get the value of `NumSamples` for these scopes, in the MATLAB window, type

```
get(sc12,'NumSamples')
```

You get a result like the following:

```
ans =
    [300]
    [300]
```

To get a list of readable properties, type `scope_object`. The property values are listed in the MATLAB window.

Note Method names are case sensitive. You must type the entire name. Property names are not case sensitive. You do not need to type the entire name as long as the characters you do type are unique for the property.

Using the Method Syntax with Scope Objects

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

- `scope_object.method_name(argument_list)`
- `scope_object_vector(index_vector).method_name(argument_list)`

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

1 In the MATLAB window, type

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

2 Alternatively, you can type

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

Acquiring Signal Data with Scopes of Type File

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

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

1 In the MATLAB window, type

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

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

2 Type

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

xPC Target adds signal `4` to the scope of type `file`. When you start the scope and application, the scope saves the signal data for signal `4` to a file, by default named `C:\data.dat`.

See “Scope of Type File” on page 3-34 in Chapter 3, “Signals and Parameters,” for a description of signal tracing with scopes of type `file`.

Advanced Data Acquisition Topics

The moment that an xPC Target scope begins to acquire data is user configurable. You can have xPC Target scopes acquire data right away, or define triggers for scopes such that the xPC Target scopes wait until they are triggered to acquire data. You can configure xPC Target scopes to start acquiring data when the following scope trigger conditions are met. These are known as trigger modes.

- `Freerun` — Starts to acquire data as soon as the scope is started (default)
- `Software` — Starts to acquire data in response to a user request. You generate a user request when you call the scope method `trigger` or the scope function `xPCScSoftwareTrigger`
- `Signal` — Starts to acquire data when a particular signal has crossed a preset level
- `Scope` — Starts to acquire data based on when another (triggering) scope starts

You can use several properties to further refine when a scope acquires data. For example, if you set a scope to trigger on a signal (`Signal` trigger mode), you can configure the scope to specify the following:

- The signal to trigger the scope (required)
- The trigger level that the signal must cross to trigger the scope to start acquiring data
- Whether the scope should trigger on a rising signal, falling signal, or either one

In the following topics, the trigger point is the sample during which the scope trigger condition is satisfied. For signal triggering, the trigger point is the sample during which the trigger signal passes through the trigger level. At the trigger point, the scope acquires the first sample. By default, scopes start acquiring data from the trigger point onwards. You can modify this behavior using the pre- and posttriggering.

- Pre-triggering — Starts to acquire data *N* moments before a trigger occurs
- Post-triggering — Starts to acquire data *N* moments after a trigger occurs

The `NumPrePostSamples` scope property controls the pre- and posttriggering operation. This property specifies the number of samples to be collected before or after a trigger event.

- If `NumPrePostSamples` is a negative number, the scope is in pretriggering mode, where it starts collecting samples before the trigger event.
- If `NumPrePostSamples` is a positive number, the scope is in a posttriggering mode, where it starts collecting samples after the trigger event.

The following topics describe two examples of acquiring data:

- "Triggering One Scope with Another Scope to Acquire Data" on page 2-13 — Describes a configuration of one scope to trigger another using the concept of pre- and posttriggering
- "Acquiring Gap-Free Data Using Two Scopes" on page 2-16 — Describes how to apply the concept of triggering one scope with another to acquire gap-free data

Triggering One Scope with Another Scope to Acquire Data

This section describes the concept of triggering one scope with another to acquire data. The description uses actual scope objects and properties to describe triggers.

The ability to have one scope trigger another, and to delay retrieving data from the second after a trigger event on the first, is most useful when data acquisition for the second scope is triggered after data acquisition for the first scope is complete. In the following explanation, Scope 2 is triggered by Scope 1.

- Assume two scopes objects are configured as a vector with the command

  ```
  sc = tg.addscope('host', [1 2]);
  ```

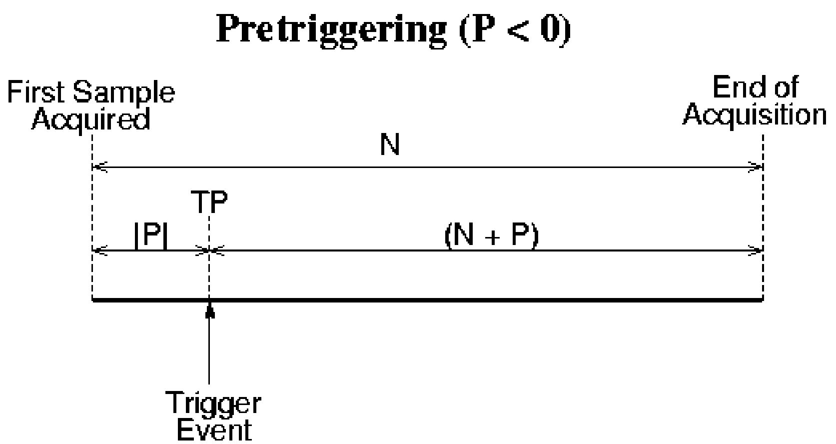
- For Scope 1, set the following values:
  - `sc(1).ScopeId = 1`
  - `sc(1).NumSamples = N`
  - `sc1.NumPrePostSamples = P`

- For Scope 2, set the following values:
  - `sc(2).ScopeId = 2`
  - `sc(2).TriggerMode = 'Scope'`
  - `sc(2).TriggerScope =1`
  - `sc(2).TriggerSample = range 0 to (N + P - 1)`

In the figures below, `TP` is the trigger point or sample where a trigger event occurs. Scope 1 begins acquiring data as described.

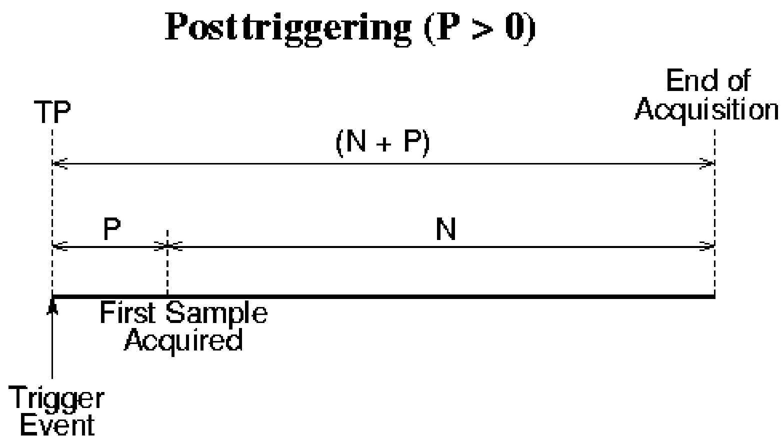
In the simplest case, where `P = 0`, Scope 1 acquires data right away.

"Pretriggering (P<0)" on page 2-14 illustrates the behavior if `P`, the value of `NumPrePostSamples`, is negative. In this case, Scope 1 starts acquiring data `|P|` samples before `TP`. Scope 2 begins to acquire data only after `TP` occurs.



Pretriggering (P<0)

"Posttriggering (P>0)" on page 2-15, illustrates the behavior if `P`, the value of `NumPrePostSamples`, is positive. In this case, Scope 1 starts acquiring data `|P|` samples after `TP` occurs.



Posttriggering (P>0)

Scope 1 triggers Scope 2 after the trigger event occurs. The following describes some of the ways you can trigger Scope 2:

- `sc(2).TriggerSample = 0` — Causes Scope 2 to be triggered when Scope 1 is triggered. `TP` for both scopes as at the same sample.
- `sc(2).TriggerSample = n > 0` — Causes `TP` for Scope 2 to be `n` samples after `TP` for Scope 1. Note that setting `sc(2).TriggerSample` to a value larger than `(N + P - 1)` does not cause an error; it implies that Scope 2 will never trigger, since Scope 1 will never acquire more than `(N + P - 1)` samples after `TP`.
- `sc(2).TriggerSample = 0 < n < (N + P)` — Enables you to obtain some of the functionality that is available with pre- or posttriggering. For example, if you have the following Scope 1 and Scope 2 settings,
  - Scope 1 has `sc(1).NumPrePostSamples = 0` (no pre- or posttriggering)
  - Scope 2 has `sc(2).TriggerSample = 10`
  - Scope 2 has `sc(2).NumPrePostSample = 0`

  The behavior displayed by Scope 2 is equivalent to having `sc(2).TriggerSample = 0` and `sc(2).NumPrePostSamples = 10`.
- `sc(2).TriggerSample = -1` — Causes Scope 2 to start acquiring data from the sample after Scope 1 stops acquiring.

Note The difference between setting `TriggerSample = (N + P - 1)`, where `N` and `P` are the parameters of the triggering scope (Scope 1) and `TriggerSample = -1` is that in the former case, the first sample of Scope 2 will be at the same time as the last sample of Scope 1, whereas in the latter, the first sample of Scope 2 will be one sample after the last sample of Scope 1. This means that in the former case both scopes acquire simultaneously for one sample, and in the latter they will never simultaneously acquire.

Acquiring Gap-Free Data Using Two Scopes

With two scopes, you can acquire gap-free data. Gap-free data is data that two scopes acquire consecutively, with no overlap. The first scope acquires data up to N, then stops. The second scope begins to acquire data at N+1. This section provides guidelines for setting up two scopes for gap-free data. This is functionality that you cannot achieve through pre- or posttriggering.

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

  ```
  sc1.TriggerSample = -1
  sc2.TriggerSample = -1
  ```

- Set the `TriggerScope` property for each scope so that each is triggered by the other. For example,

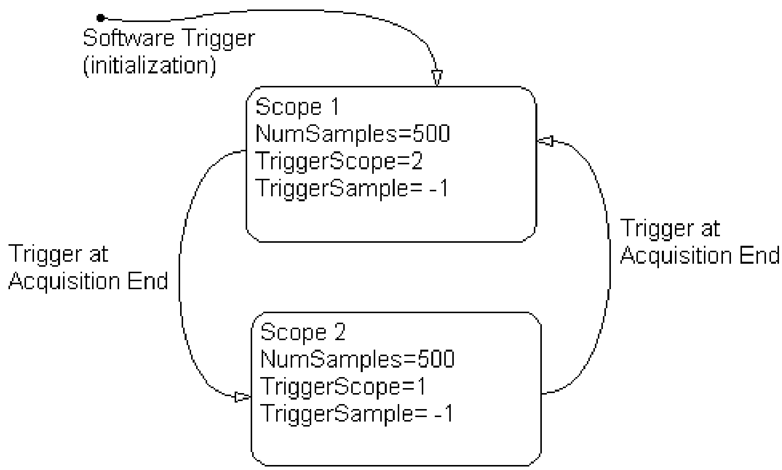
  ```
  sc1.TriggerScope=2
  sc2.TriggerScope=1
  ```

- Set the `NumSamples` property for each scope. For example,

  ```
  sc1.NumSamples=500
  sc2.NumSamples=500
  ```

- Set the `TriggerMode` property for one of the scopes to `'Software'`. You must do this to start the data acquisition. Otherwise, each scope waits for the other to finish acquiring data, and never starts. In “Acquisition of Gap-Free Data” on page 2-17, the `TriggerMode` property of Scope 1 is set to `'Software'`. This allows Scope 1 to be software triggered to acquire data when it receives the command `sc1.trigger`.
- Both the scopes receive exactly the same signals, in other words, the signals you want to retrieve.

"Acquisition of Gap-Free Data" on page 2-17, illustrates how the scopes trigger one another.



Acquisition of Gap-Free Data

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

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

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

3

Signals and Parameters

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

Monitoring Signals

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

In addition to signal monitoring, xPC Target enables you to monitor test-pointed Stateflow® states through the xPC Target Explorer and MATLAB command-line interfaces. You designate data or a state in a Stateflow diagram as a test point. This makes it observable during execution. See the Stateflow and Stateflow Coder user's guide documentation for details. You can work with Stateflow states as you do xPC Target signals, such as monitoring or plotting Stateflow states. (See "Monitoring Stateflow States" on page 3-6 for details.)

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

This section has the following topics:

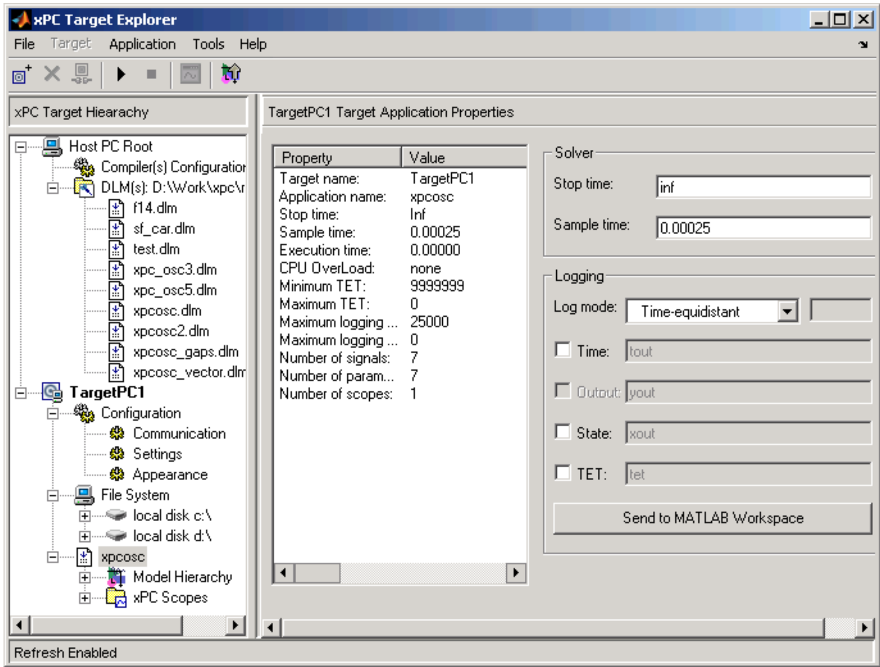
Signal Monitoring with xPC Target Explorer

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

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

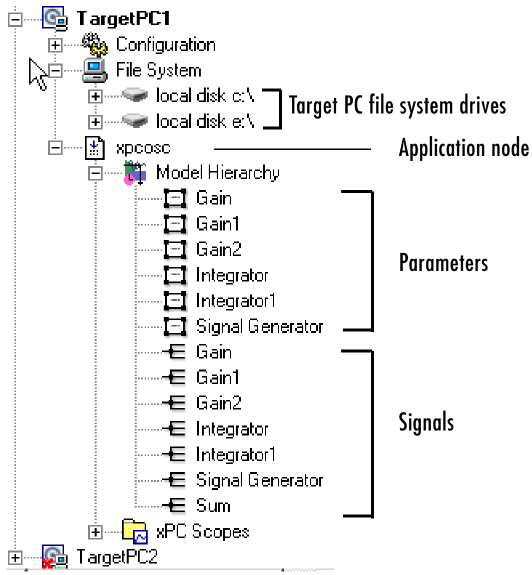
   The target PC **Target Application Properties** pane appears.

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



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

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

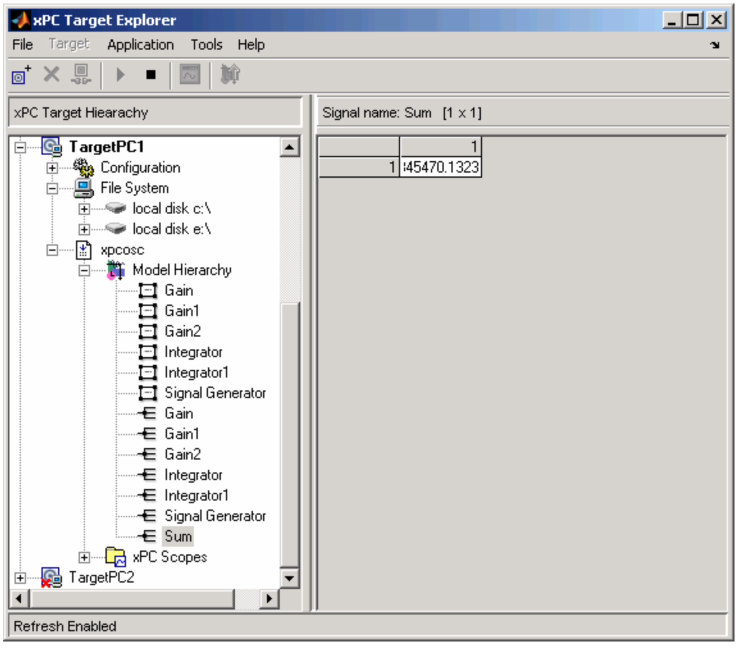


The `Model Hierarchy` node can have up to three types of nodes: subsystems ( ) (including their signals and parameters), parameters ( ), and signals ( ). Only xPC Target tunable parameters and signals of the application, as represented in the Simulink model, appear in the `Model Hierarchy` node. This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

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

4 To get the value of a signal, select the signal in the `Model Hierarchy` node.

The value of the signal is shown in the right pane. For example,



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

The application starts running.

Signal Monitoring with MATLAB

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

1 To get a list of signals, type either

```
set(tg, 'ShowSignals', 'On') or tg.ShowSignals='On'
```

The latter command causes the MATLAB window to display a list of the target object properties for the available signals. For example, the signals for the model xpc_osc3.mdl are shown below.

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

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

```
tg.getsignal(1)
```

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

```
ans=
     3.731
```

See also “Signal Tracing with MATLAB” on page 3-26.

Monitoring Stateflow States

This procedure uses the model sf_car.mdl as an example. It describes one way to test-point Stateflow states for monitoring.

1 In the MATLAB window, type

```
sf_car
```

2 In the Simulink window, and from the **Simulation** menu, click **Configuration Parameters**.

The Configuration Parameters dialog box is displayed for the model.

3 Click the **Real-Time Workshop®** node.

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

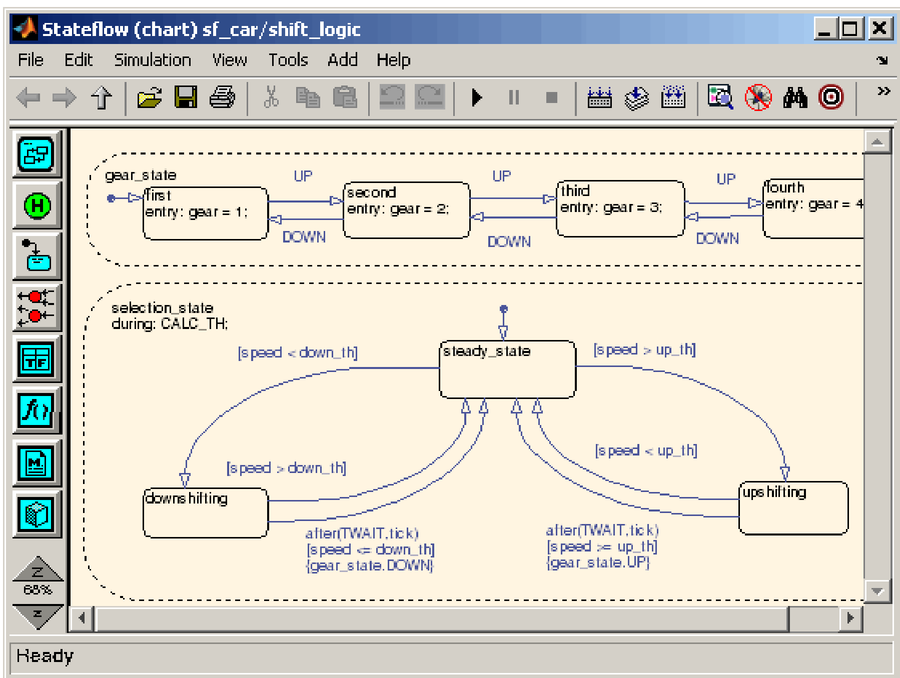
4 To build a basic target application, in the **Target selection** section, click the **Browse** button at the **RTW system target file** list. Click `xpctarget.tlc`, and then click **OK**.

5 As necessary, you can click the **xPC Target options** node and continue to make changes.

6 When you are done, click **OK**.

7 In the `sf_car` model, double-click the `shift_logic` chart.

The `shift_logic` chart is displayed.



8 In the chart, select **Tools -> Explore**.

   The Model Explorer is displayed.

9 In the Model Explorer, expand `sf_car`.

10 Expand `gear_state`, then select `first`.

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

12 Click **Apply**.

13 Build and download the `sf_car` target application to the target PC.

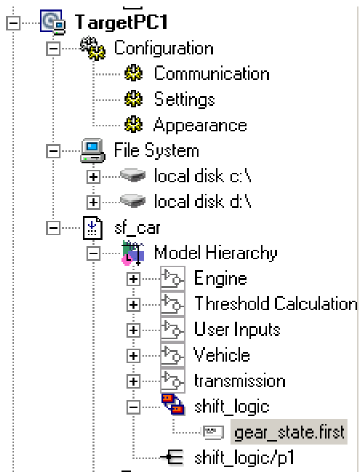
You can now view the states with the xPC Target Explorer or MATLAB.

Monitoring Stateflow States with xPC Target Explorer

This topic assumes that you have already test-pointed Stateflow states (see "Monitoring Stateflow States" on page 3-6 if you have not).

1 If the xPC Target Explorer is not yet started, start it now and connect it to the target PC that has the downloaded `sf_car` target application.

2 To view the test point in the xPC Target Explorer, expand the Model Hierarchy node and navigate to `shift_logic`. The test point `gear_state.first` appears like any other signal in the hierarchy, as follows:



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

Monitoring Stateflow States with MATLAB

This topic assumes that you have already test-pointed Stateflow states (see "Monitoring Stateflow States" on page 3-6 if you have not).

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

```
tg=xpc
```

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

`set(tg, 'ShowSignals', 'On'); tg` or `tg.ShowSignals='On'`

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

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

```
shift_logic:gear_state.first
```

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

Signal Tracing

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

- “Signal Tracing with xPC Target Explorer” on page 3-11 — Use the xPC Target Explorer to create and run scopes that are displayed on the host PC.
- “Signal Tracing with MATLAB” on page 3-26 — Use the MATLAB Command Window to create scopes and scope objects.
- “Signal Tracing with xPC Target Scope Blocks” on page 3-33 — Use scopes of type `host` to trace signal data triggered by an event while your target application is running.
- “Signal Tracing with a Web Browser” on page 3-34 — Use Microsoft Explorer or Netscape Navigator to view signals.

Signal tracing differs from signal logging. With signal logging you can only look at signals after a run is finished and the log of the entire run is available. For information on signal logging, see “Signal Logging” on page 3-36.

Signal Tracing with xPC Target Explorer

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

- “Creating Scopes” on page 3-12 — Create scopes on the host PC and target PC to visualize the data.
- “Adding Signals to Scopes” on page 3-16 — Add signals to the scopes and start the scopes.
- “Stopping Scopes” on page 3-20 — Stop the scopes.

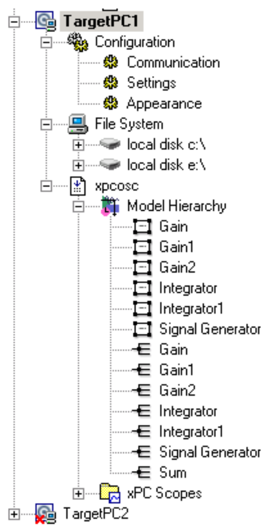
You can add or remove signals from scopes of type `target` or `host` while the scope is either stopped or running. For scopes of type `file`, you must stop the scope first before adding or removing signals.

Creating Scopes

1 In xPC Target Explorer, ensure that your `xpcosc` application is still running.

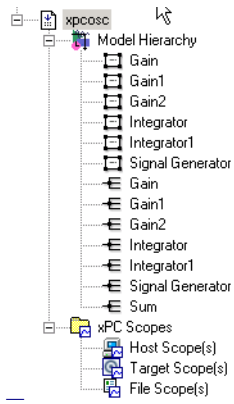
2 To get the list of signals in the target application, expand the `Model Hierarchy` node under the target application.

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



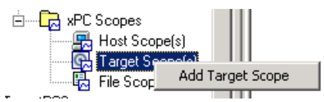
3 To get the list of scope types you can have in the target application, expand the `xPC Scopes` node below the application node.

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



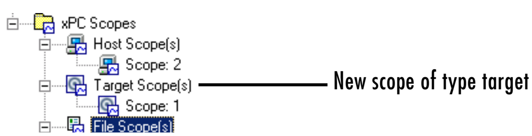
4 To create a scope to display on the target PC, right-click the `Target Scopes` node under the `xPC Scopes` node.

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



5 Select **Add Target Scope**.

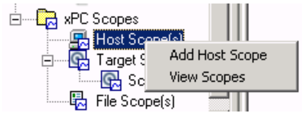
A scope node appears under `Target Scopes`. In this example, the new scope is labeled `Scope 1`.



You can add other scopes, including those of type `host` and `file`, up to ten each.

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

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

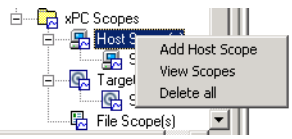


7 Select **Add Host Scope**.

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

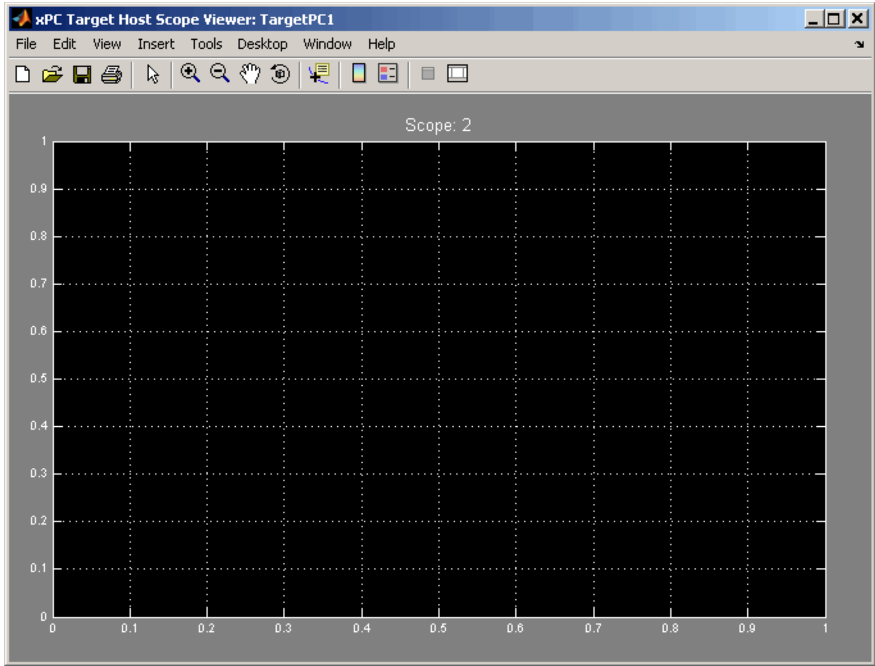
8 To visualize the host scope on the host PC, right-click `Host Scopes` from the `xPC Scopes` node.

A drop-down list appears.



9 Select **View Scopes**.

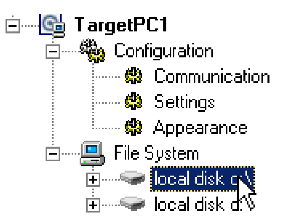
The **xPC Target Host Scope Viewer** appears. Note that the signals you add to the scope will appear at the top right of the viewer.



10 To list the properties of the scope object `Scope 2`, select the xPC Target Explorer tab to return to that window, and left-click `Scope 2`. (Note that you can configure the docking view using the MATLAB docking feature.)

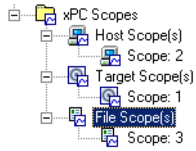
The scope properties are shown in the rightmost pane.

11 To create a scope to acquire signal data into a file on the target PC file system, in the **xPC Target Hierarchy** pane, select the target PC drive and subdirectory in which you create the file for the signal data. For example, select `local disk c:\` to create a file in the top-level directory of the C drive.



12 Right-click the `File Scopes` node under the `xPC Scopes` node.

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

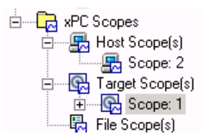


You next task is to add signals to the scopes. The following procedure assumes that you have added scopes to the target PC and host PC.

Adding Signals to Scopes

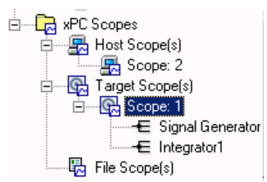
1 In the xPC Target Explorer window, add signals to the target PC scope, `Scope 1`. For example, to add signals `Integrator1` and `Signal Generator`, right-click each signal and select **Add to Scopes**. From the **Add to Scopes** list, select `Scope 1`. (Note that you can also drag the signal to the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

The `Scope 1` node is shown with a plus sign.

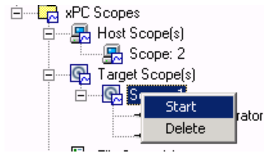


2 Expand the `Scope 1` node.

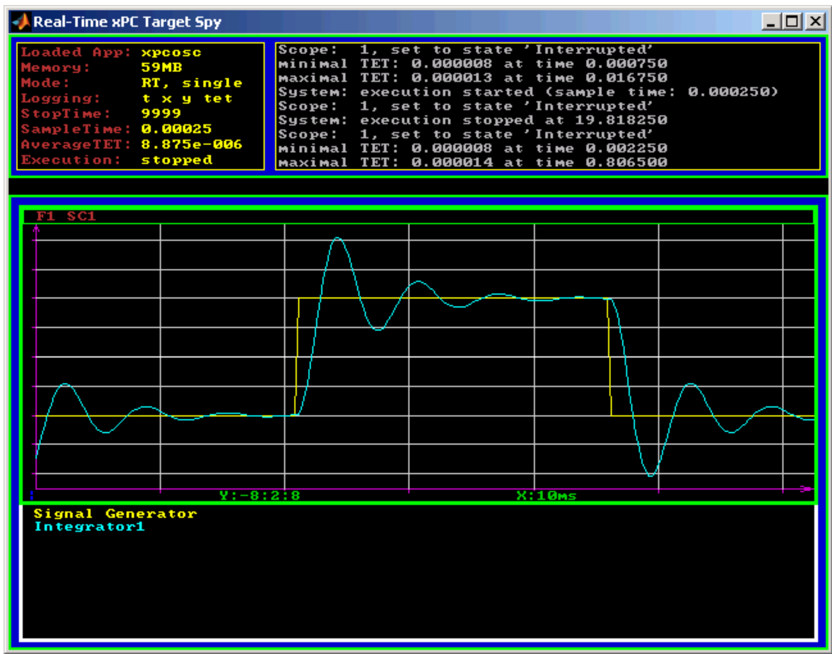
The `Scope 1` signals are displayed.



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



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

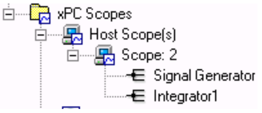


4 Add signals to the host PC scope. For example, to add signals `Integrator1` and `Signal Generator`, right-click each signal and select **Add to Scopes**. From the **Add to Scopes** list, select `Scope 2`.

The `Scope 2` node is shown with a plus sign.

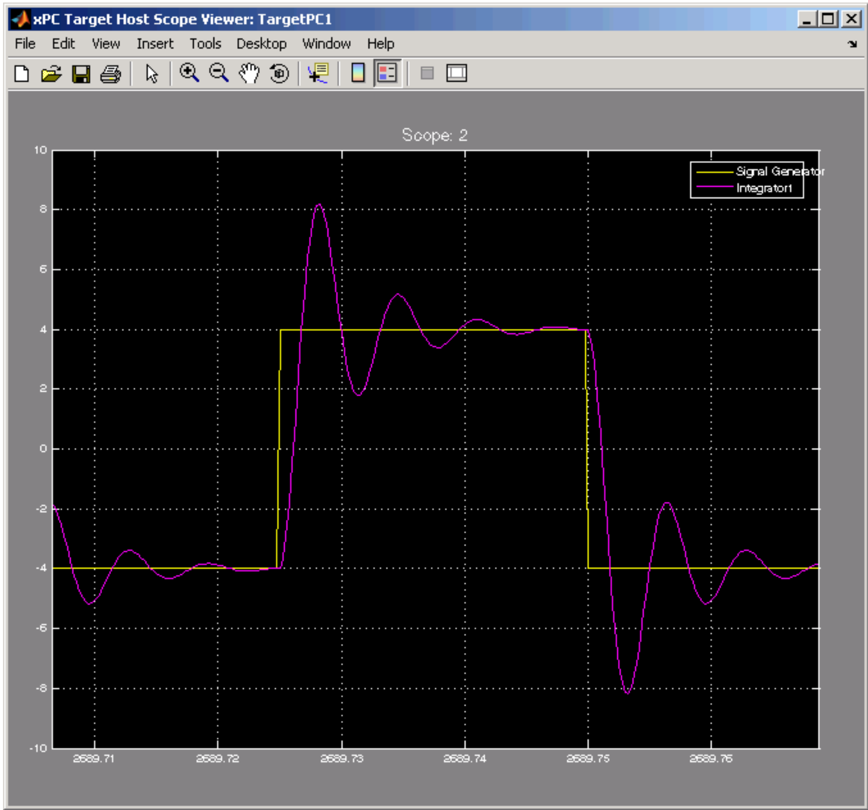
5 Expand the `Scope 2` node.

   The `Scope 2` signals are displayed.

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

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

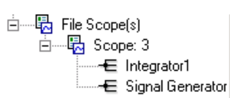


7 Add signals to the scope of type `file`. For example, to add signals `Integrator1` and `Signal Generator`, right-click each signal and select **Add to Scopes**. From the **Add to Scopes** list, select `Scope 3`.

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

8 Expand the `Scope 3` node.

The `Scope 3` signals are displayed.



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

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

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

   For scopes of type `file`, xPC Target saves data to a file on the target PC flash disk.

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

Stopping Scopes

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

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

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

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

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

```
System: execution stopped
minimal TET: 0.0000006 at time 0.001250
maximal TET: 0.0000013 at time 75.405500
```

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

If you have scopes of type `file`, you can copy the file that contains the signal data from the target PC to the host PC. See "Copying Files to the Host PC" on page 3-24.

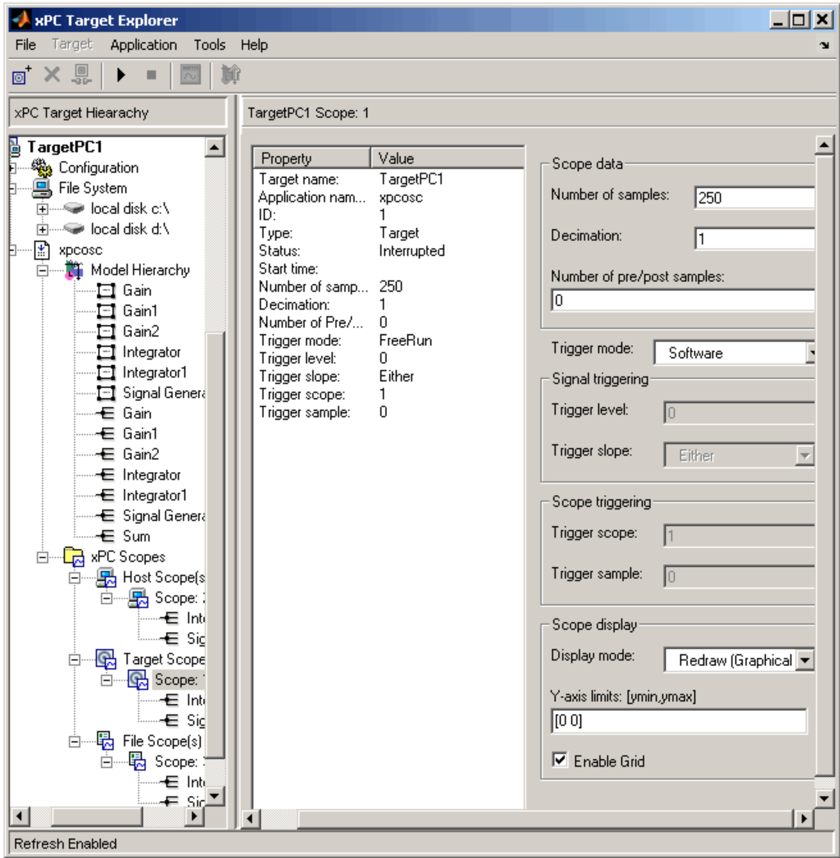
Software Triggering Scopes

You can set up a scope such that only a user can trigger the scope. This section assumes that you have added a scope to your target application (see "Creating Scopes" on page 3-12) and that you have added signals to that scope ("Adding Signals to Scopes" on page 3-16).

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

   The properties pane for that scope is displayed.

2 From the **Trigger mode** list, select `Software`.



3 Start the scope and target application.

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

5 Select `Trigger`.

Configuring the Host Scope Viewer

You can customize your host scope viewer. This section assumes that you have added a host scope to your target application, started the host scope viewer, and added signals `Integrator1` and `Signal Generator` (see “Creating Scopes” on page 3-12 and “Adding Signals to Scopes” on page 3-16). These viewer settings are per scope.

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

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

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

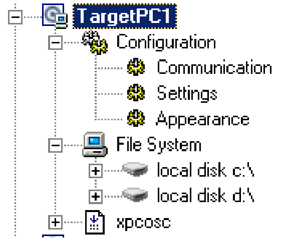
Copying Files to the Host PC

From **xPC Target Explorer**, you can download target PC files from the target PC to the host PC. This is most useful when you want to examine the data from the scope data file.

1 In xPC Target Explorer, expand the target PC node associated with the target PC file system you want to access. For example, expand `TargetPC1`.

2 Under `TargetPC1`, expand the `File System` node.

   Nodes representing the drives on the target PC are displayed.



3 Expand the node of the drive that contains the file you want. For example, `local disk: c:\`.

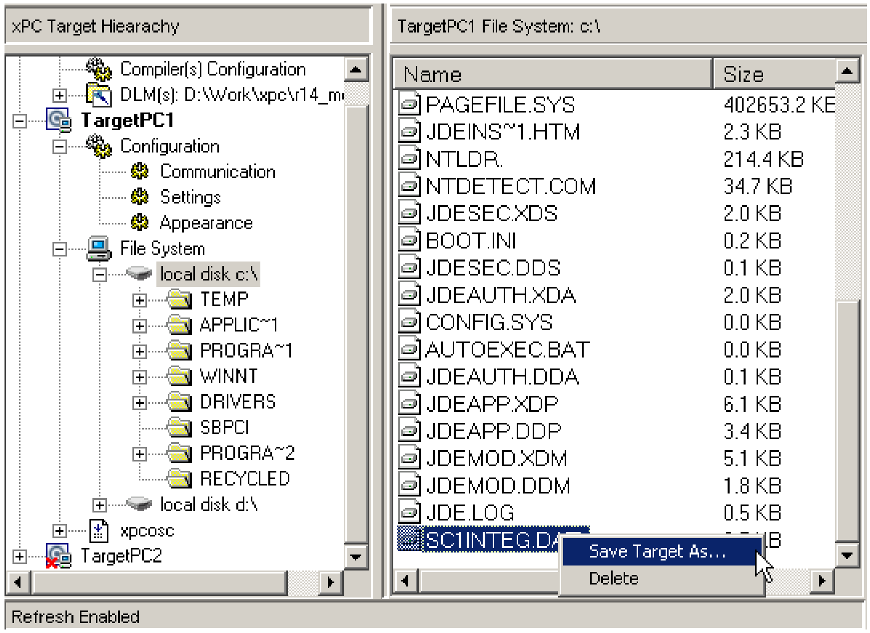
4 Select the directory that contains the file you want. For example, select the node labeled `local disk: c:\`.

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

5 In the right pane, right-click the file you want to copy to the host PC. For example, right-click `SC1INTEG.DAT`.

   A context-sensitive menu is displayed.

6 Select **Save Target As**.



A file Save As dialog is displayed.

7 Choose the directory to contain the signal data file. If you want, you can also save the file under a different name.

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

You can examine the contents of the signal data file. See "Retrieving a File from the Target PC to the Host PC" on page 7-7 in Chapter 7, "Working with Target PC Files and File Systems."

Deleting Files from the Target PC

From **xPC Target Explorer** on the host PC, you can delete the scope data file on the target PC file system. Use the same procedure as described in "Copying Files to the Host PC" on page 3-24, but select **Delete** instead of **Save Target As**. **xPC Target Explorer** removes the selected file from the target PC file system.

Signal Tracing with MATLAB

Creating a scope object allows you to select and view signals using the scope methods. This section describes how to trace signals using xPC Target functions instead of using the xPC Target graphical user interface. This procedure assumes that you have assigned `tg` to the appropriate target PC.

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

Using the MATLAB interface, you can trace signals with

- Host or target scopes (see "Signal Tracing with MATLAB and Scopes of Type Target" on page 3-26 for a description of signal tracing with target scopes)
- Scopes of type `file` (see "Signal Tracing with MATLAB and Scopes of Type File" on page 3-29)

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

Signal Tracing with MATLAB and Scopes of Type Target

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

1 Start running your target application. Type any of

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

The target PC displays the following message.

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

2 To get a list of signals, type either

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

or

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

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

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

For more information, see "Signal Monitoring with MATLAB" on page 3-6.

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

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

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

```
sc1
```

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

```
xPC Scope Object

   Application          = xpcosc
   ScopeId              = 1
   Status               = Interrupted
   Type                 = Target
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = -1
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 1
   TriggerSample        = -1
   Mode                 = Redraw (Graphical)
```

```
YLimit               = Auto
Grid                 = On
Signals              = no Signals defined
```

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

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

The target PC displays the following messages.

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

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

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

```
+sc1 or sc.start or start(sc1)
```

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

7 Stop the scope. Type either

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

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

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

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

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

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

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

Signal Tracing with MATLAB and Scopes of Type File

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

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

---

1 Create an xPC Target application that works with scopes of type `file`. Type

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

2 To get a list of signals, type either

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

or

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

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

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

For more information, see “Signal Monitoring with MATLAB” on page 3-6.

3 Start running your target application. Type

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

The target PC displays the following message:

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

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

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

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

```
sc2
```

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

```
xPC Scope Object

   Application          = xpcosc
   ScopeId              = 2
   Status               = Interrupted
   Type                 = File
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerScope         = 2
   TriggerSample        = 0
   TriggerSignal        = -1
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   Signals              = no Signals defined
   StartTime            = -1.000000
   FileName             = unset
```

```
    Mode                  = Lazy
    WriteSize             = 512
    AutoRestart           = off
```

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

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

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

The target PC displays the following messages.

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

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

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

```
+sc2 or sc2.start or start(sc2)
```

The target PC displays the following message.

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

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

```
Application          = xpcosc
    ScopeId              = 2
    Status               = Pre-Acquiring
    Type                 = File
    NumSamples           = 250
    NumPrePostSamples    = 0
    Decimation           = 1
    TriggerMode          = FreeRun
```

```
TriggerScope         = 2
TriggerSample        = 0
TriggerSignal        = 0
TriggerLevel         = 0.000000
TriggerSlope         = Either
StartTime            = NaN
Signals              = 0  : Integrator1
                       1  : Signal Generator
StartTime            = NaN
FileName             = c:\sc2Integ.dat
Mode                 = Lazy
WriteSize            = 512
AutoRestart          = off
```

8 Stop the scope. Type

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

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

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

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

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

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

```
System: execution stopped
minimal TET: 0.00006 at time 0.004250
maximal TET: 0.000037 at time 14.255250
```

To access the contents of the signal data file that the xPC Target scope of type `file` creates, use the xPC Target file system object (`xpctarget.fs`) from the host PC MATLAB window. To view or examine the signal data, you can use the `readxpcfile` utility in conjunction with the `plot` function. For further details on the `xpctarget.fs` file system object and the `readxpcfile` utility, see Chapter 7, “Working with Target PC Files and File Systems.”

Signal Tracing with xPC Target Scope Blocks

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

Note xPC Target supports ten scopes of type `target` and `host`, and eight scopes of type `file`, for a maximum of 28 scopes.

Scope of Type Host

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

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

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

Scope of Type Target

For a scope of type `target`, logged data (`sc.Data`, `sc.Time`, and `sc.StartTime`) is not accessible over the command-line interface on the host PC. This is because the scope object status (`sc.Status`) is never set to `Finished`. Once the scope completes one data cycle (time to collect the number of samples), the scope engine automatically restarts the scope. If you create a scope object, for example, `sc = getscopes(tg,1)` for a scope of type `target`, 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 PC while the data is displayed on the target PC, you need to define a second scope object with type `host`. Then you need to synchronize the acquisitions of the two scope objects by setting **TriggerMode** for the second scope to `'Scope'`.

Scope of Type File

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

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

Signal Tracing with 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 PC you can use the scopes page to add, remove, and control scopes on the target PC:

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

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

2 Click the **Add Scope** button.

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

   To create a scope of type `host`, use the drop-down list next to the **Add Scope** button to select `Host`. This item is set to `Target` by default.

3 Click the **Edit** button.

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

4 Click the **Add Signals** button.

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

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

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

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

When a host scope is stopped (**Scope State** is set to `Interrupted`) or finishes one cycle of acquisition (**Scope State** is set to `Finished`), a button called **Get Data** appears on the page. If you click this button, the scope data is retrieved in comma separated variable (CSV) format. The signals in the scope are spread across columns, and each row corresponds to one sample of acquisition. The first column always corresponds to the time each sample was acquired.

Note If **Scope State** is set to `Interrupted`, the scope was stopped before it could complete a full cycle of acquisition. Even in this case, the number of rows in the CSV data will correspond to a full cycle. The last few rows (for which data was not acquired) will be set to `0`.

Signal Logging

Signal logging is the process for acquiring signal data during a real-time run, stopping the target application, and then transferring the data to the host PC for analysis. You can plot and analyze the data, and later save it to a disk. xPC Target signal logging samples at the base sample time. If you have a model with multiple sample rates, add xPC Target scopes to the model to ensure that signals are sampled at their appropriate sample rates.

Note xPC Target does not support logging data with decimation.

This section includes the following topics:

- "Signal Logging with xPC Target Explorer" on page 3-36 — Use an xPC Target Scope block in your Simulink model to log signal data triggered by an event.
- "Signal Logging with MATLAB" on page 3-38 — Use Outport blocks in your Simulink model to log data to a target object in the MATLAB workspace.
- "Signal Logging with a Web Browser" on page 3-42 — Use Microsoft Internet Explorer or Netscape Navigator to log data to a text file.

Signal Logging with xPC Target Explorer

You plot the outputs and states of your target application to observe the behavior of your model, or to determine the behavior when you vary the input signals and model parameters.

This procedure uses a model named `xpc_osc4.mdl` as an example and assumes you have created a target application and downloaded it to the target PC. The `xpc_osc4.mdl` is the same as `xpc_osc3.mdl` with the xPC Target Scope block removed. See "xPC Target Application" in Chapter 3 in the xPC Target getting started documentation.

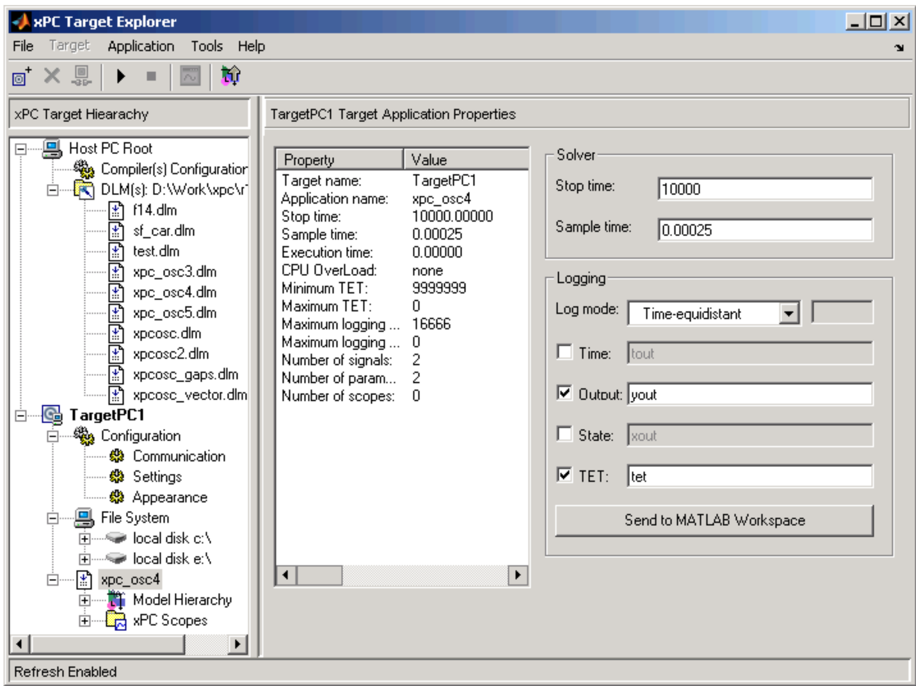
Note To use the xPC Target Explorer for signal logging, you need to add an Outport block to your Simulink model, and you need to activate logging on the **Data Import/Export** pane in the Configuration Parameters dialog.

---

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

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

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

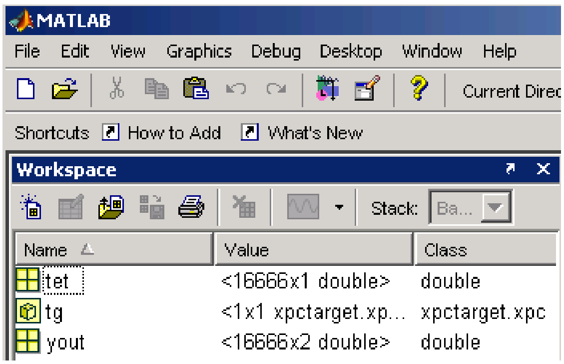


3 Start the target application. For example, in the **Target Hierarchy** pane, select the `xpc_osc4` target application, then select **Start**.

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

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

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



You can examine or otherwise manipulate the data.

Signal Logging with MATLAB

You plot the outputs and states of your target application to observe the behavior of your model, or to determine the behavior when you vary the input signals and model parameters.

Time, states, and outputs — Logging the output signals is possible only if you add Outport blocks to your Simulink model before the build process, and in the **Configuration Parameters Data Import/Export** node, select the **Save to workspace** check boxes. See “Entering Parameters for the Outport Blocks” in Chapter 3 of the xPC Target getting started documentation.

Task execution time — Plotting the task execution time is possible only if you select the **Log Task Execution Time** check box in the **Configuration Parameters xPC Target options** tab. This check box is selected by default. See “Adding an xPC Target Scope Block” in Chapter 3 of the xPC Target getting started documentation.

All scopes copy the last `N` samples from the log buffer to the target object logs (`tg.TimeLog`, `tg.OutputLog`, `tg.StateLog`, and `tg.TETLog`). xPC Target calculates the number of samples `N` for a signal as the value of **Signal logging buffer size in doubles** divided by the number of logged signals (1 time, 1 task execution time (TET), outputs, states).

After you run a target application, you can plot the state and output signals. This procedure uses the Simulink model `xpc_osc3.mdl` as an example, and assumes you have created and downloaded the target application for that model. It also assumes that you have assigned `tg` to the appropriate target PC.

1 In the MATLAB window, type

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

The target application starts and runs until it reaches the final time set in the target object property `tg.StopTime`.

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

2 Plot the signals from the Outport block and the states. In the MATLAB window, type

```
plot(tg.TimeLog,tg.Outputlog)
```

Values for the logs are uploaded to the host PC from the target application on the target PC. If you want to upload part of the logs, see the target object method `getlog` on page 14-44.

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

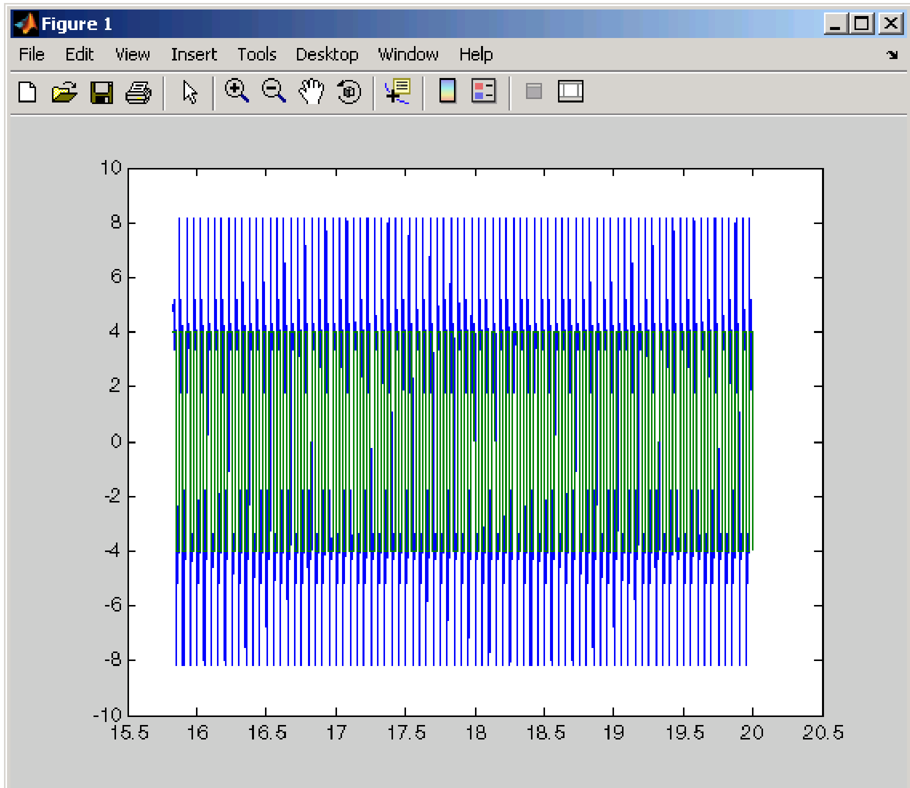


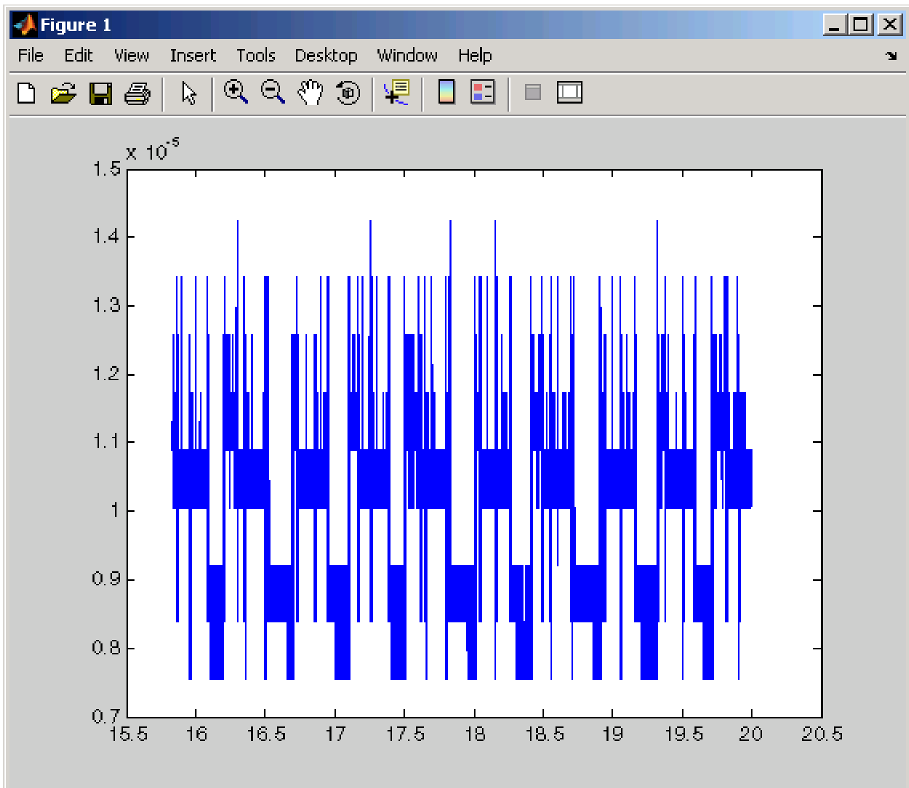
Figure 1
File Edit View Insert Tools Desktop Window Help
10
8
6
4
2
0
-2
-4
-6
-8
-10
15.5 16 16.5 17 17.5 18 18.5 19 19.5 20 20.5

3 In the MATLAB window, type

```
plot(tg.TimeLog,tg.TETLog)
```

Values for the task execution time (TET) log are uploaded to the host PC from the target PC. If you want to upload part of the logs, see the target object method `getlog` on page 14-44.

The plot shown below is the result of a real-time run.



The TET is the time to calculate the signal values for the model during each sample interval. If you have subsystems that run only under certain circumstances, plotting the TET would show when subsystems were executed and the additional CPU time required for those executions.

4 In the MATLAB window, type either

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

MATLAB displays the following information about the average task execution time.

```
ans =

  5.7528e-006
```

The percentage of CPU performance is the average TET divided by the sample time.

Signal Logging with a Web Browser

When you stop the model execution, another section of the Web browser interface appears that enables you to download logging data. This data is in comma-separated value (CSV) format. This format can be read by most spreadsheet programs and also by MATLAB using the `csvread` function.

This section of the Web browser interfaces appears only if you have enabled data logging, and buttons appear only for those logs (states, output, and TET) that are enabled. If time logging is enabled, the first column of the CSV file is the time at which data (states, output, and TET values) was acquired. If time logging is not enabled, only the data is in the CSV file, without time information.

You analyze and plot the outputs and states of your target application to observe the behavior of your model, or to determine the behavior when you vary the input signals.

Time, states, and outputs — Logging the output signals is possible only if you add Outport blocks to your Simulink model before the build process, and in the **Configuration Parameters Data Import/Export** node, select the **Save to workspace** check boxes. See "Entering Parameters for the Outport Blocks" in Chapter 3 in the xPC Target getting started documentation.

Task execution time — Logging the task execution time is possible only if you select the **Log Task Execution Time** check box in the **Configuration Parameters xPC Target options** node. This check box is selected by default. See "Entering Parameters for an xPC Target Scope Block" in Chapter 3 in the xPC Target getting started documentation.

Parameter Tuning and Inlining Parameters

By default, xPC Target lets you change parameters in your target application while it is running in real time.

You can also improve overall efficiency by inlining parameters. xPC Target supports the Real-Time Workshop inline parameters functionality (see the using Real-Time Workshop documentation for further details on inlined parameters). By default, this functionality makes all parameters nontunable. If you want to make some of the inlined parameters tunable, you can do so through the Model Parameter Configuration dialog (see "Inlined Parameters" on page 3-56.)

This section includes the following topics:

- "Parameter Tuning with xPC Target Explorer" on page 3-44 — Use the xPC Target Explorer to change block parameters in your target application.
- "Parameter Tuning with MATLAB" on page 3-47 — Use the MATLAB Command Window and target objects in your MATLAB workspace to change target application parameters.
- "Parameter Tuning with Simulink External Mode" on page 3-50 — Connect your Simulink model to your target application, and change target application parameters by changing Simulink block parameters.

Note Opening a dialog box for a source block causes Simulink to pause. While Simulink is paused, you can edit the parameter values. You must close the dialog box to have the changes take effect and allow Simulink to continue.

- "Parameter Tuning with a Web Browser" on page 3-53 — Connect your target application to a Web browser with the target application running on a target PC connected to a network.
- "Inlined Parameters" on page 3-56 — Inline parameters and specify that some parameters can be tunable.

Parameter Tuning with xPC Target Explorer

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

You can download parameters to the target application while it is running or between runs. This feature lets you change parameters in your target application without rebuilding the Simulink model. You cannot use xPC Target Explorer to change tunable source block parameters while a simulation is running.

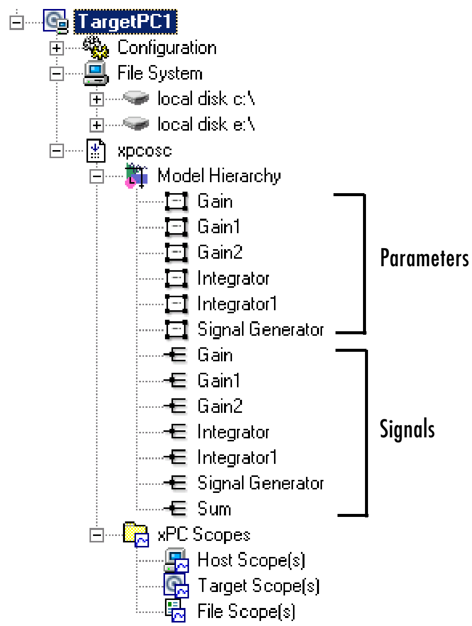
After you download a target application to the target PC, you can change block parameters using xPC Target Explorer. This procedure uses the Simulink model `xpcosc.mdl` as an example, and assumes you have created and downloaded the target application for that model.

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

2. Select **Start**.

3. To get the list of parameters in the target application, expand the `Model Hierarchy` node under the target application.

   The `Model Hierarchy` expands to show the elements in the Simulink model.



The model hierarchy only shows blocks that have tunable parameters.

4 Select the parameter of the signal you want to edit. For example, select `Gain`.

The right pane displays the block parameters dialog for Gain. There is one parameter, **Gain**, for this block. The current value of the **Gain** parameter is displayed.

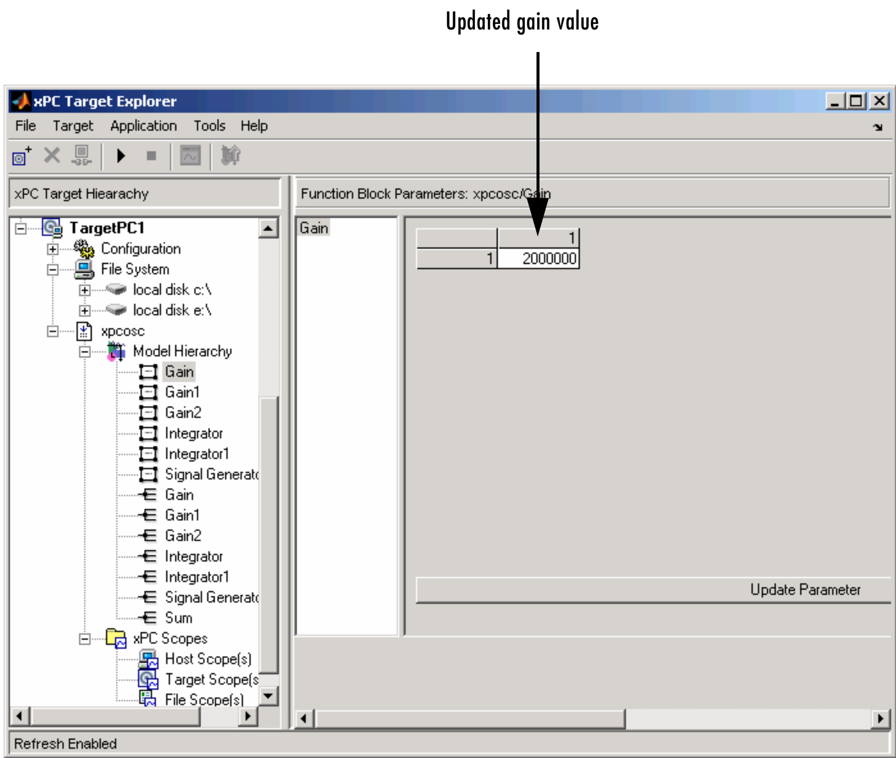
5 Double-click the box that contains the gain value.

The box becomes editable.

6 Enter a new value for the gain.

7 Press the **Enter** key.

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



If there is a scope, the plot frame then updates the signals after running the simulation with the new parameter value.

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

The target application on the target PC stops running.

Parameter Tuning with MATLAB

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

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

After you download a target application to the target PC, you can change block parameters using xPC Target functions. This procedure uses the Simulink model `xpcosc.mdl` as an example, and assumes you have created and downloaded the target application for that model. It also assumes that you have assigned `tg` to the appropriate target PC.

1. In the MATLAB window, type

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

   The target PC displays the following message.

   ```
   System: execution started (sample time: 0.001000)
   ```

2. Display a list of parameters. Type either

   ```
   set(tg,'ShowParameters','on') or tg.ShowParameters='on'
   ```

   and then type

   ```
   tg
   ```

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

   ```
   ShowParameters = on
   ```

```
Parameters =
```

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

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

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

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

```
ans =

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

If there is a scope, the plot frame then updates the signals after running the simulation with the new parameters.

4 Stop the target application. In the MATLAB window, type

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

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

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

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

Resetting Target Application Parameters to Previous Values

You can reset parameters to preceding target object property values by using xPC Target methods on the host PC. The `setparam` method returns a structure that stores the parameter index, the previous value, and the new value. If you expect to want to reset parameter values, set the `setparam` method to a variable. This variable points to a structure that stores the parameter index and the old and new parameter values for it.

1 In the MATLAB window, type

```
pt=setparam(tg,5,800)
```

The `setparam` method returns a result like:

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

2 To reset to the previous values, type

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

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

Parameter Tuning with Simulink External Mode

You use Simulink external mode to connect your Simulink block diagram to your target application. The block diagram becomes a graphical user interface to your target application. You set up Simulink in external mode to establish a communication channel between your Simulink block window and your target application.

In Simulink external mode, wherever you change parameters in the Simulink block diagram, Simulink downloads those parameters to the target application while it is running. This feature lets you change parameters in your program without rebuilding the Simulink model to create a new target application.

Note Opening a dialog box for a source block causes Simulink to pause. While Simulink is paused, you can edit the parameter values. You must close the dialog box to have the changes take effect and allow Simulink to continue.

After you download your target application to the target PC, you can connect your Simulink model to the target application. This procedure uses the Simulink model `xpcosc.mdl` as an example, and assumes you have created and downloaded the target application for that model.

1 In the Simulink window, and from the **Simulation** menu, click **External**.

   A check mark appears next to the menu item **External**, and Simulink external mode is activated.

2 In the Simulink block window, and from the **Simulation** menu, click **Connect to target**.

All of the current Simulink model parameters are downloaded to your target application. This downloading guarantees the consistency of the parameters between the host model and the target application.

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

```
ExtM: Updating # parameters
```

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

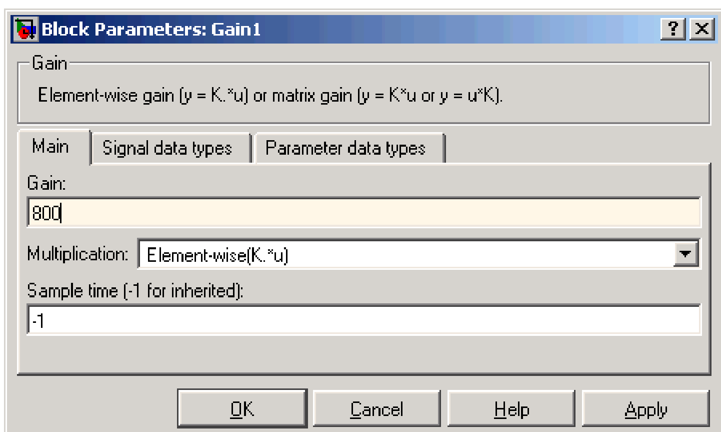
```
+tg or tg.start or start(tg)
```

The target application begins running on the target PC, and the target PC displays the following message:

```
System: execution started (sample time: 0.000250)
```

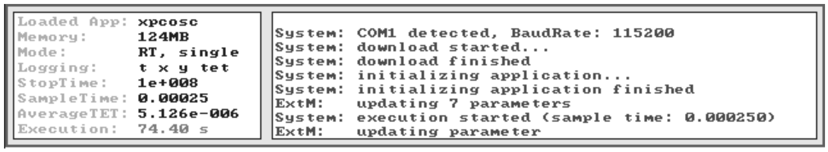
4 From the Simulation block diagram, double-click the block labeled **Gain1**.

The Block Parameters: Gain1 parameter dialog box opens.



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

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



6 From the **Simulation** menu, click **Disconnect from Target**.

The Simulink model is disconnected from the target application. Now, if you change a block parameter in the Simulink model, there is no effect on the target application. Connecting and disconnecting Simulink works regardless of whether the target application is running or not.

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

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

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

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

Parameter Tuning with a Web Browser

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

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

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

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

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

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

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

   The new parameter values are uploaded to the target application.

Saving and Reloading Application Parameters with MATLAB

After you have a set of target application parameter values that you are satisfied with, you can save those values to a file on the target PC. You can then later reload these saved parameter values to the same target application. You can save parameters from your target application while the target application is running or between runs. This feature lets you save and restore parameters in your target application without rebuilding the Simulink model. You save and restore parameters with the target object methods `saveparamset` and `loadparamset`.

The procedures assume that

- You have a target application object named `tg`.
- You have assigned `tg` to the appropriate target PC.
- You have a target application downloaded on the target PC.

- You have parameters you would like to save for reuse. See
  - "Parameter Tuning with MATLAB" on page 3-47
  - "Parameter Tuning with Simulink External Mode" on page 3-50
  - "Parameter Tuning with a Web Browser" on page 3-53

Saving the Current Set of Target Application Parameters

To save a set of parameters to a target application, use the `saveparamset` method. The target application can be stopped or running.

1 Identify the set of parameter values you want to save.

2 Select a descriptive filename to contain these values. For example, use the model name in the filename. You can only load parameter values to the same target application from which you saved the parameter values.

3 In the MATLAB window, type either

   `tg.saveparamset('xpc_osc4_param1')` or
   `saveparamset(tg,'xpc_osc4_param1')`

   xPC Target creates a file named `xpcosc4_param1` in the current directory of the target PC, for example, `C:\xpcosc4_param1`.

For a description of how to restore parameter values to a target application, see "Loading Saved Parameters to a Target Application" on page 3-54. For a description of how to list the parameters and values stored in the parameter file, see "Listing the Values of the Parameters Stored in a File" on page 3-55.

Loading Saved Parameters to a Target Application

To load a set of saved parameters to a target application, use the `loadparamset` method. You must load parameters to the same model from which you save the parameter file. If you load a parameter file to a different model, the behavior is undefined.

This section assumes that you have a parameters file saved from an earlier run of `saveparamset` (see "Saving the Current Set of Target Application Parameters" on page 3-54).

1 From the collection of parameter value files on the target PC, select the one that contains the parameter values you want to load.

2 In the MATLAB window, type either

```
tg.loadparamset('xpc_osc4_param1') or
loadparamset(tg,'xpc_osc4_param1')
```

xPC Target loads the parameter values into the target application.

For a description of how to list the parameters and values stored in the parameter file, see “Listing the Values of the Parameters Stored in a File” on page 3-55.

Listing the Values of the Parameters Stored in a File

To list the parameters and their values, load the file for a target application, then turn on the `ShowParameters` target object property.

This section assumes that you have a parameters file saved from an earlier run of `saveparamset` (see “Saving the Current Set of Target Application Parameters” on page 3-54).

1 Ensure that the target application is stopped. For example, type

```
tg.stop
```

2 Load the parameter file. For example, type

```
tg.loadparamset('xpc_osc4_param1');
```

3 Display a list of parameters. For example, type

```
tg.ShowParameters='on';
```

and then type

```
tg
```

The MATLAB window displays a list of parameters and their values for the target object.

Inlined Parameters

This procedure describes how you can globally inline parameters for a model, then specify which of these parameters you still want to be tunable. It assumes that you are familiar with how to build target applications (if you are not, read the xPC Target getting started documentation first). After you have performed this procedure, you will able to tune these parameters.

- "Tuning Inlined Parameters with xPC Target Explorer" on page 3-58
- "Tuning Inlined Parameters with MATLAB" on page 3-60

The following procedure uses the Simulink model `xpcosc.mdl` as an example.

1 In the MATLAB Command Window, type

```
xpcosc
```

The model is displayed in the Simulink window.

2 Select the blocks of the parameters you want to make tunable. For example, this procedure makes the signal generator's amplitude parameter tunable. Use the variable `A` to represent the amplitude.

3 Double-click the Signal Generator block and enter `A` for the `Amplitude` parameter. Click **OK**.

4 In the MATLAB Command Window, assign a constant to that variable. For example, type

```
A = 4
```

The value is displayed in the MATLAB workspace.

5 In the Simulink window, from the **Simulation** menu, click **Configuration Parameters**.

The Configuration Parameters dialog is displayed for the model.

6 Click the **Optimization** node.

7 In the rightmost pane, select the Inline parameters check box.

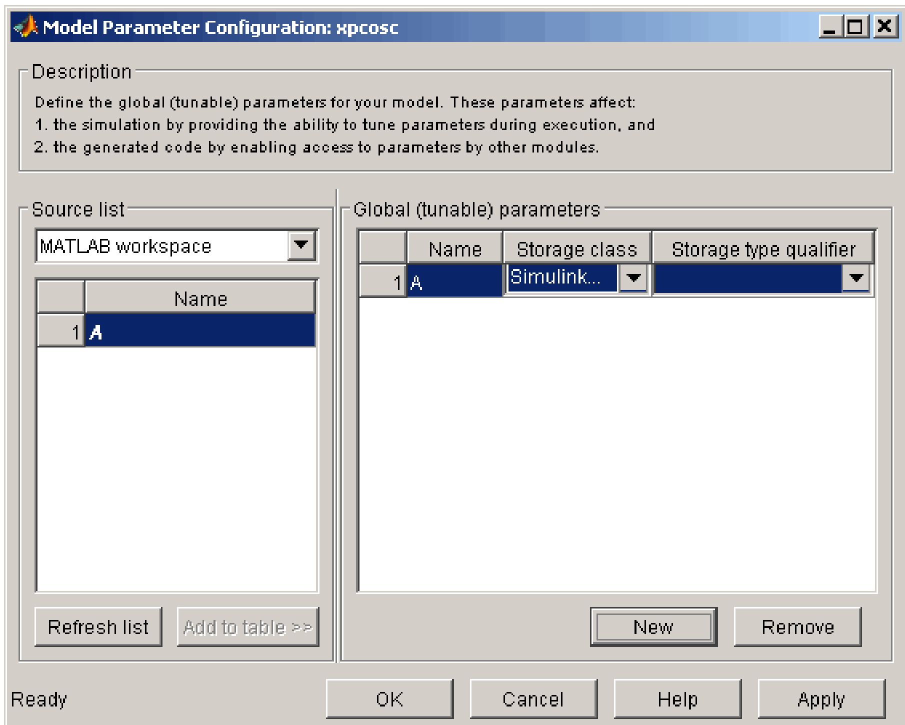
The **Configure** button is enabled.

8 Click the **Configure** button.

The Model Parameter Configuration dialog is displayed. Note that the MATLAB workspace contains the constant you assigned to `A`.

9 Select the line that contains your constant and click **Add to table**.

The Model Parameters Configuration dialog appears as follows.



If you have more global parameters you want to be able to tune, add them also.

10 If you want, increase the model stop time, or set it to `inf`.

11 When you are finished, click **Apply**, then **OK**, and save the model. For example, save it as `xpc_osc5.mdl`.

12 Build and download the model to your target PC.

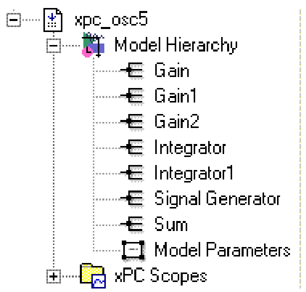
You can next use xPC Target Explorer or MATLAB to work with the tunable parameters.

Tuning Inlined Parameters with xPC Target Explorer

This procedure describes how you can tune inlined parameters through the xPC Target Explorer. It assumes that you have built and downloaded the model from the topic "Inlined Parameters" on page 3-56 to the target PC. It also assumes that the model is running.

1 If you have not yet started xPC Target Explorer, do so now. Be sure it is connected to the target PC to which you downloaded the `xpc_osc5` target application.

2 To get the list of tunable inlined parameters in the target application, expand the target application node, then expand the `Model Hierarchy` node under the target application node.



Note that the `Model Hierarchy` node displays a list of signals and an object called `Model Parameters`. `Model Parameters` contains the list of tunable inlined parameters.

3 To display the tunable parameters, select `Model Parameter`.

The constant `A` and its value are shown in the right pane.

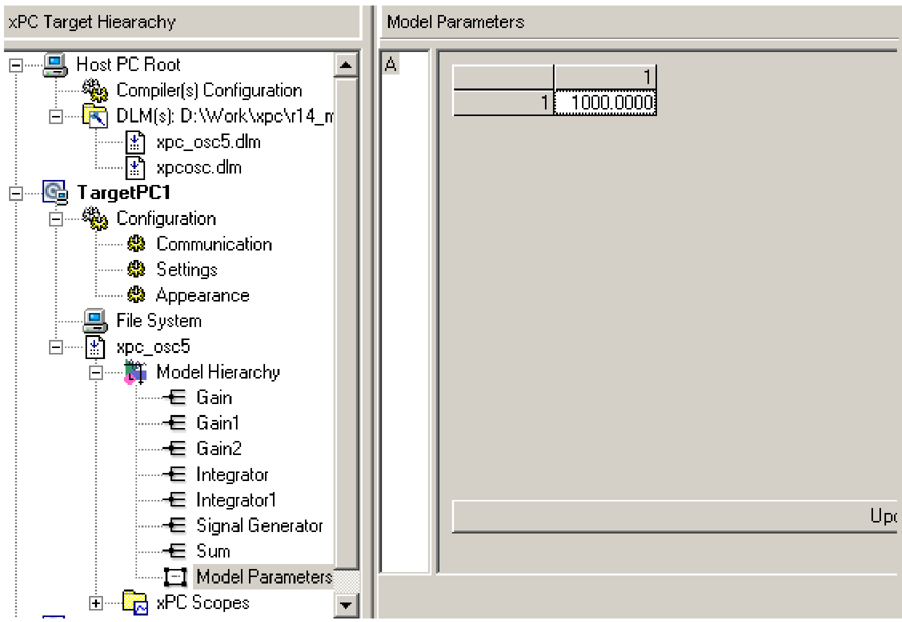
4 Double-click the box that contains the tunable parameter `A`.

The box becomes editable.

5 Enter a new value for the parameter.

6 Press the **Enter** key.

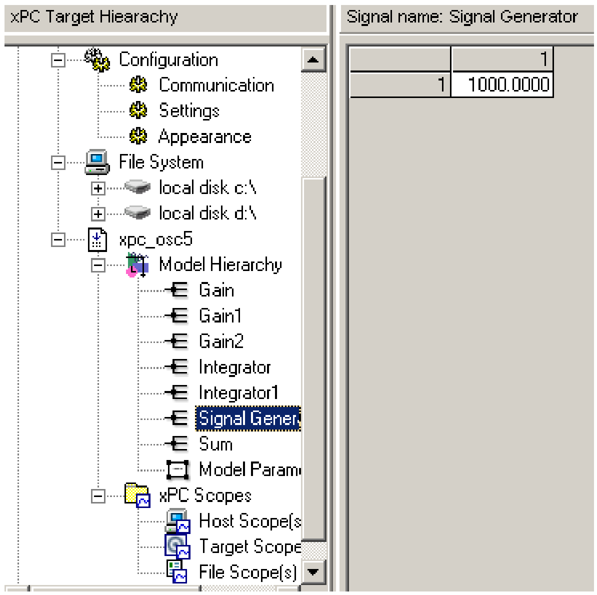
The box is updated and the **Update Parameter** button becomes active.



7 To apply the new value, press the **Update Parameter** button.

8 To verify the updated value, select the signal object associated with `A`. For example, select `Signal Generator`.

The value of `Signal Generator` is shown in the right pane.



9 Stop the target application.

Tuning Inlined Parameters with MATLAB

This procedure describes how you can tune inlined parameters through MATLAB. It assumes that you have built and downloaded the model from the topic "Inlined Parameters" on page 3-56 to the target PC. It also assumes that the model is running.

You can tune inline parameters using a parameter ID similar to the way that you use conventional parameters.

- Use the `getparamid` function to get the ID of the inlined parameter you want to tune. For the `block_name` parameter, leave a blank `('')`.
- Use the `setparam` function to set the new value for the inlined parameter.

1 Save the following code in an M-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 M-file. For example, type

```
change_inlineA
```

3 To see the new parameter value, type

```
tg.showparameters='on'
```

The `tg` object information is displayed, including the parameters lines:

```
NumParameters = 1
ShowParameters = on
Parameters=INDEX  VALUE  TYPE   SIZE   PARAMETER  NAM  BLOCKNAME
             0      100    DOUBLE Scalar A
```

4

Embedded Option

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

Introduction

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

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

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

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

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

xPC Target Embedded Option Modes

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

- DOSLoader — Use this mode of operation to start the kernel on the target PC from a 3.5 inch disk, flash disk, or a hard disk. After the target PC boots with the kernel, it waits for the host computer to download a real-time application. You can control the target application from either the host PC or the target PC. See "DOSLoader Mode Overview" on page 4-4 for further details.
- StandAlone — Use this mode to load the target PC with both the xPC Target kernel and a target application. Like DOSLoader mode, this mode of operation can start the kernel on the target PC from 3.5 inch disk, flash disk, or hard disk. After starting the kernel on the target PC, StandAlone mode can also automatically start the target application that you loaded with the kernel. Thus, this configuration provides complete stand-alone operation. StandAlone mode eliminates the need for a host PC and allows you to deploy real-time applications on target PCs. See "StandAlone Mode Overview" on page 4-6 for further details.

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

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

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

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

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

This section includes the following topics:

- "DOSLoader Mode Overview" on page 4-4
- "StandAlone Mode Overview" on page 4-6
- "Restrictions" on page 4-8

DOSLoader Mode Overview

The primary purpose of DOSLoader mode is to allow you to boot from devices other than a 3.5 inch disk drive. The following summarizes the sequence of events for DOSLoader mode. For a detailed step-by-step procedure, see "DOSLoader Target Setup" on page 4-12.

1. Format a 3.5 inch disk.

2. Copy a version of DOS onto this disk and insert this DOS disk into the host PC 3.5 inch disk drive.

3. In the host PC MATLAB Command Window, type `xpcexplr`.

4. In the xPC Target Explorer **xPC Target Hierarchy** pane, select a target PC `Configuration` node.

5. From the **Target boot mode** list, select `DOSLoader`.

6. Create a boot disk. The boot disk will contain the following files:

   - DOS files — Provide your own copy of DOS to boot the target PC. For example, you can acquire DOS from FreeDOS.

     The MathWorks has tested xPC Target with the following: FreeDOS Beta 8 ("Nikita") distribution, MS-DOS (6.0 or higher), PC DOS, and Caldera OpenDOS.

- autoexec.bat — xPC Target version of this file that calls the xpcboot.com executable to boot the xPC Target kernel.
- checksum.dat — xPC Target version of this file that optimizes boot disk creation.
- *.rtb — This file contains the xPC Target kernel. It also contains, as applicable, specifications such as serial or TCP/IP communications and the IP address of the target PC.
- xpcboot.com — Contains the xPC Target boot executable. This file executes an xPC Target application and executes the *.rtb file.

7 Move the boot disk to the target PC.

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

9 Boot the target PC.

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

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

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

StandAlone Mode Overview

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

The following summarizes the sequence of events for StandAlone mode. For a detailed step-by-step procedure, see "Stand-Alone Target Setup" on page 4-17.

1. Format a 3.5 inch disk.

2. Copy a version of DOS onto this disk and insert this DOS disk into the host PC 3.5 inch disk drive.

3. From the host PC MATLAB window, type `xpcexplr`.

4. In the xPC Target Explorer **xPC Target Hierarchy** pane, select a target PC `Configuration` node.

5. From the **Target boot mode** list, select `StandAlone`.

6. Select and build a model.

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

7. Copy the contents of `model_name_emb` to the DOS disk. The disk should now contain the following files:

   - DOS files — Provide your own copy of DOS to boot the target PC. For example, you can acquire DOS from FreeDOS.

     The MathWorks has tested xPC Target with the following: FreeDOS Beta 8 ("Nikita") distribution, MS-DOS (6.0 or higher), PC DOS, and Caldera OpenDOS.
   - `*.rtb` — This file contains the xPC Target kernel. It also contains, as applicable, specifications such as serial or TCP/IP communications and the IP address of the target PC.

- `xpcboot.com` — Contains the xPC Target boot executable. This file executes an xPC Target application and executes the `*.rtb` file.
- `autoexec.bat` — xPC Target version of this file that calls the `xpcboot.com` executable to boot the xPC Target kernel.

8 Move the boot disk to the target PC.

9 Set up the target PC boot device such as a 3.5 inch disk, flash disk, or a hard disk drive. Transfer the contents of the boot disk to the target PC boot device.

10 Boot the target PC.

When you boot the target PC, the target PC loads DOS, which then calls the xPC Target `autoexec.bat` file to start the xPC Target kernel (`*.rtb`) and associated target application. If you set up the boot device to run the xPC Target `autoexec.bat` file upon start-up, the target application starts executing as soon as possible. The xPC Target application executes entirely in protected mode using the 32-bit flat memory model.

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

With StandAlone mode, the target PC does not communicate with the host PC. If you want to track signals on the target PC monitor, your target PC hardware configuration needs to include a monitor. To trace signals, you must add xPC Target scopes to the application before you build and transfer it to the target PC. See “Adding Target Scope Blocks to Stand-Alone Applications” on page 4-18.

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

Restrictions

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

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

To satisfy these restrictions,

- Do not use additional memory managers like `emm386` or `qemm`.
- Avoid any utilities that attempt to load in high memory (for example, `himem.sys)`. If the target PC DOS environment does not use a `config.sys` file or memory manager entries in the `autoexec.bat` file, there should be no problems when running `xpcboot.com`.
- Ensure that the `xpcexplr` **TargetMouse** option setting is consistent with your hardware. Some PC hardware might use an RS-232 port for the mouse, while others use a PS2 mouse. If a mouse is not required in your application, select `None` as your setting for the **TargetMouse**. Choosing this setting helps prevent problems.

Embedded Option Setup

This section includes the following topics:

Updating the xPC Target Environment

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

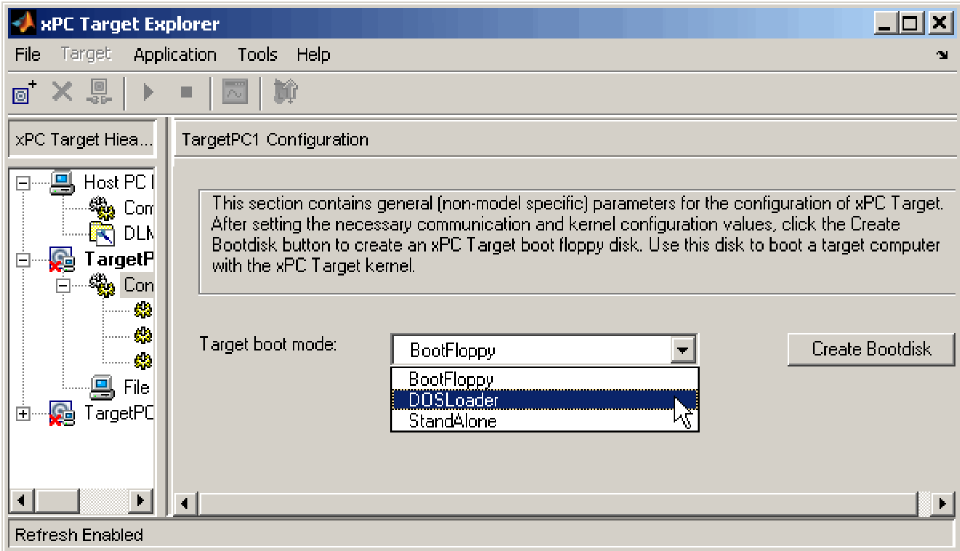
It is assumed that the xPC Target environment is already set up and working properly with the xPC Target Embedded Option enabled. If you have not already done so, confirm this now.

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

```
xpcexplr
```

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

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



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

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

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

For more detailed information about how to use the **xPC Target Explorer** window, see "xPC Target Explorer" in Chapter 2 in the xPC Target getting starting documentation.

Creating a DOS System Disk

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

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

```
sys A:
```

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

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

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

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

DOSLoader Target Setup

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

This section includes the following topics:

- "Updating Environment Properties and Creating a Boot Disk" on page 4-12 — Select `DOSLoader` mode in the xPC Target Explorer window.
- "Copying the Kernel to Flash Memory" on page 4-14 — Copy the xPC Target kernel to the flash disk on the target PC and then start the kernel running.
- "Creating a Target Application for DOSLoader Mode" on page 4-16 — Create, download, and run a target application from a host PC.

Updating Environment Properties and Creating a Boot Disk

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

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

1 On the host computer, start MATLAB.

2 In the MATLAB Command Window, type

```
xpcexplr
```

The **xPC Target Explorer** window opens.

3 In the xPC Target Explorer **xPC Target Hierarchy** pane, select a target PC `Configuration` node.

4 From the **Target boot mode** list, select `DOSLoader`.

5 Click **Create BootDisk**.

A message box opens with the following message.

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

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

The files `checksum.dat`, `xpcsgo1.rtb` (serial) or `xpctgo1.rtb` (TCP/IP), `xpcboot.com`, and `autoexec.bat` are copied to the disk.

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

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

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

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

```
xpcboot xxx.rtb
```

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

7 If you want to boot the target PC from the 3.5 inch disk,

   a Remove the 3.5 inch disk from the host PC.

   b Put that disk into the target PC disk drive.

c Reboot the target PC. The DOS is booted from the target boot disk and the `autoexec.bat` files, resulting in the automatic execution of the xPC Target loader. From this point onward, the CPU runs in protected mode and DOS is discarded.

Otherwise, if you want to boot the target PC from flash memory instead of the 3.5 inch disk, see "Copying the Kernel to Flash Memory" on page 4-14 for a description of how to copy the kernel to flash memory. The same procedure works with flash disks and other boot devices.

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

Copying the Kernel to Flash Memory

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

After you create boot disk with the kernel files on a host PC, you can copy the kernel files from the 3.5 inch boot disk to the flash disk. See "Updating Environment Properties and Creating a Boot Disk" on page 4-12.

1 If there is a 3.5 inch disk in the external disk drive, remove it. On the target PC, press the **Reset** button.

2 Halt the boot process and bring up the DOS prompt. For example, if you see the message for selecting the operating system to start, select `Microsoft Windows`.

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

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

4 Type

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

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

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

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

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

---

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

The sequence of calls during the boot process is

a `C:\autoexec.bat`

b `C:\work\autoexec.bat`

c `C:\work\xpcboot.com`

d `C:\work\xpcsgo1.rtb`

Creating a Target Application for DOSLoader Mode

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

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

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

   ```
   xpc_osc3
   ```

   A Simulink window opens with the model.

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

3. Real-Time Workshop and xPC Target create a target application and download it to your target PC.

Stand-Alone Target Setup

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

- "Updating Environment Properties" on page 4-17 — Select StandAlone mode in the xPC Target Explorer window.
- "Adding Target Scope Blocks to Stand-Alone Applications" on page 4-18 — Add target scope blocks to models to monitor signal data.
- "Creating a Kernel/Target Application" on page 4-21 — On the host PC, create a stand-alone application.
- "Copying the Kernel/Target Application to Flash Disk" on page 4-22 — Copy the combined xPC Target kernel and target application to the flash disk on the target PC.

Updating Environment Properties

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

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

1. On the host computer, start MATLAB.

2. In the MATLAB window, type

   ```
   xpcexplr
   ```

   The **xPC Target Explorer** window opens.

3. In the xPC Target Explorer **xPC Target Hierarchy** pane, select a target PC `Configuration` node.

4. From the **Target boot mode** list, select `DOSLoader`.

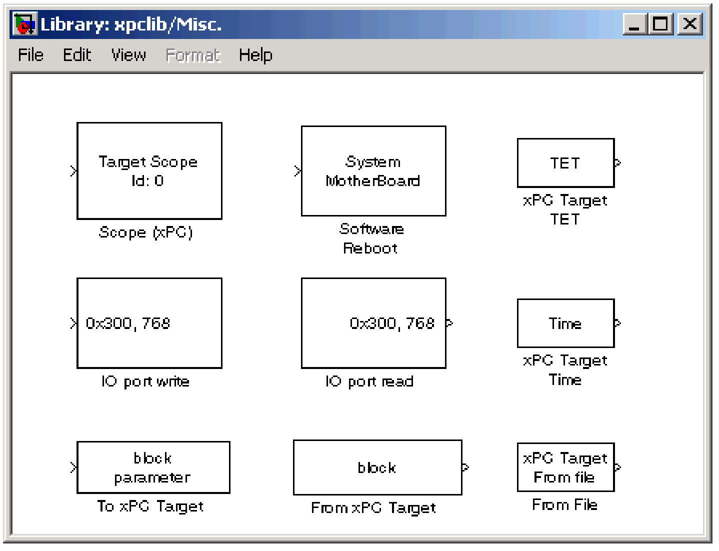
5 From the **Target boot mode** list, choose `StandAlone`.

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

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

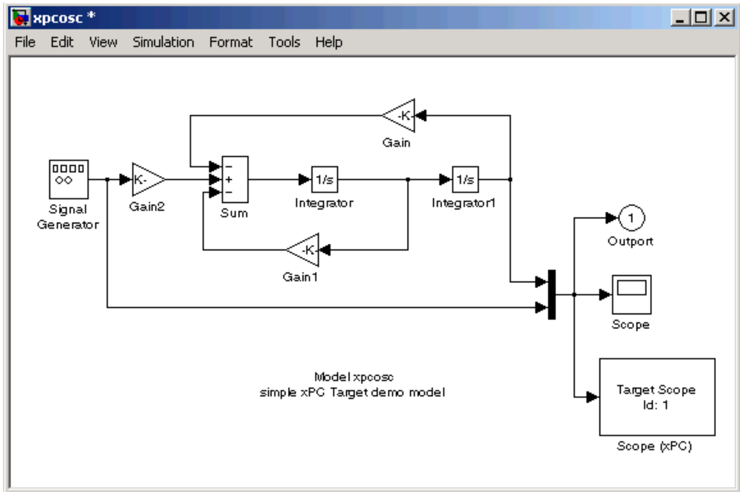
Adding Target Scope Blocks to Stand-Alone Applications

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

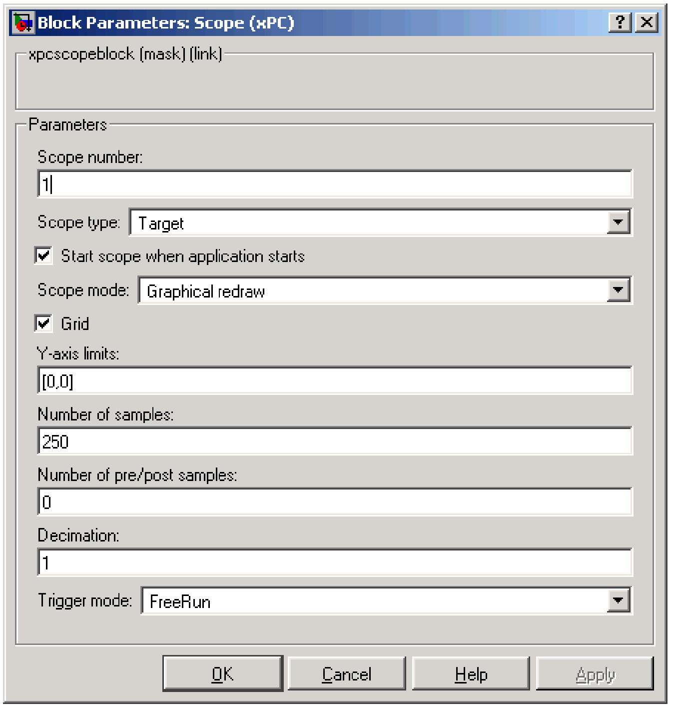


To add a **Scope (xPC)** block to a Simulink model,

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



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



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

3 Ensure that the **Scope type** field is `Target` or `File`.

4 Save the model.

Your next task is to create a kernel/target application. See "Creating a Kernel/Target Application" on page 4-21.

Creating a Kernel/Target Application

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

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

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

   ```
   xpc_osc3
   ```

   A Simulink window opens with the model.

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

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

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

3. Copy the preceding files to a formatted 3.5 inch disk.

4. If you want to boot the target PC from the 3.5 inch disk,

   a. Remove the 3.5 inch disk from the host PC.

   b. Put that disk into the target PC disk drive.

c Reboot the target PC. DOS is booted from the target boot disk and the `autoexec.bat` files, resulting in the automatic execution of the xPC Target loader. From this point onward, the CPU runs in protected mode and DOS is discarded.

If you want to boot the target PC from flash memory instead of the 3.5 inch disk, see "Copying the Kernel to Flash Memory" on page 4-14 for a description of how to copy the kernel to flash memory. The same procedure works with flash disks and other boot devices.

Copying the Kernel/Target Application to Flash Disk

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

After you build a stand-alone application on a host PC, you can copy files from a 3.5 inch disk to the flash disk. If you have not already copied the necessary files to a 3.5 inch disk, see "Creating a Kernel/Target Application" on page 4-21.

1 If there is a 3.5 inch disk in the target PC external disk drive, remove it. On the target PC, press the **Reset** button.

2 Halt the boot process and bring up the DOS prompt. For example, if you see the message for selecting the operating system to start, select `Microsoft Windows`.

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

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

4 Type

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

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

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

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

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

---

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

The sequence of calls during the boot process is

a `C:\autoexec.bat`

b `C:\work\autoexec.bat`

c `C:\work\xpcboot.com`

d `C:\work\<application>.rtb`

8 On the target PC keyboard, press the spacebar.

A command line opens on the target PC screen.

For a complete list of target PC commands, see Chapter 6, “Using the Target PC Command-Line Interface.”.

5

Software Environment and Demos

The xPC Target environment defines the connections and communication between the host and target computers. It also defines the build process for a real-time application. You can define the xPC Target environment through either the MATLAB interface or the xPC Target GUI environment.

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

Using Environment Properties and Functions

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

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

To enter properties specific to your model and its build procedure, see "Entering the Real-Time Workshop Parameters" on page 3-42. These properties are saved with your Simulink model.

Getting a List of Environment Properties

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

1. In the MATLAB window, type

   ```
   setxpcenv
   ```

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

2. Type

   ```
   getxpcenv
   ```

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

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

Changing Environment Properties with xPC Target Explorer

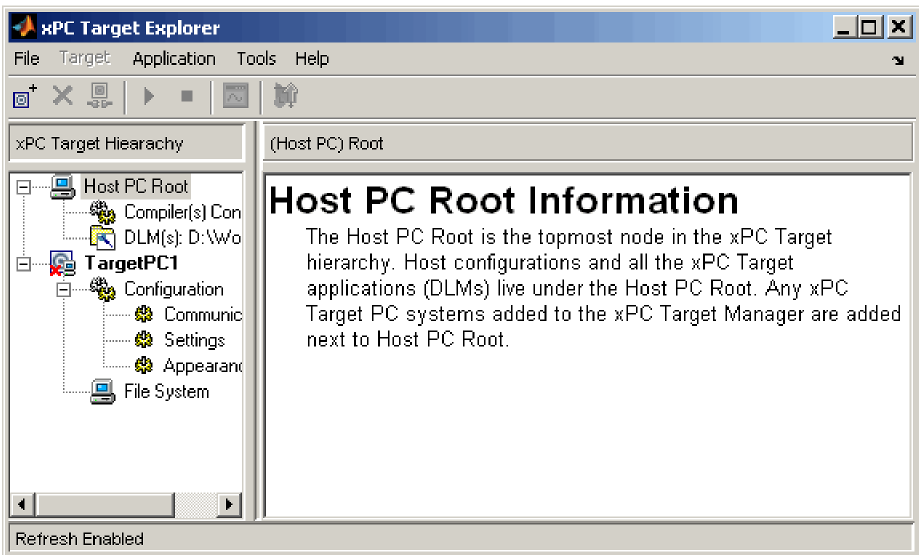
xPC Target lets you define and change environment properties. These properties include the path to the C/C++ compiler, the host PC COM port, the logging buffer size, and many others. Collectively these properties are known as the xPC Target environment.

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

1 In the MATLAB window, type

```
xpcexplr
```

MATLAB opens the **xPC Target Explorer** window.



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

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

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

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

The `Configuration` node under the `Target PC` node has the property **Target boot mode**. If your license does not include the xPC Target Embedded Option, the **Target boot mode** box is grayed out, with `BootFloppy` as your only selection. With the xPC Target Embedded Option, you have the additional choices of `DOSLoader` and `StandAlone`.

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

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

To change environment properties for target PCs, see "Configuring Environment Parameters for Target PCs" on page 5-4.

Configuring Environment Parameters for Target PCs

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

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

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

---

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

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

2 Select `Settings`.

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

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

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

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

4 From the **Maximum model size** list, select either `1 MB`, `4 MB`, or `16 MB`. Choosing the maximum model size reserves the specified amount of memory on the target PC for the target application. The remaining memory is used by the kernel and by the heap for data logging.

5 By default, the **Support secondary IDE controller** check box is not selected. Select this check box only if you want to use the disks connected to a secondary IDE controller. If you do not have disks connected to the secondary IDE controller, do not select this check box.

6 In the **xPC Target Hierarchy**, select `Appearance`.

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

7 From the **Target scope** list, select either `Enabled` or `Disabled`. The property **Target scope** is set by default to `Enabled`. If you set **Target scope** to `Disabled`, the target PC displays information as text. To use all the features of the target scope, you also need to install a keyboard and mouse on the target PC.

8 Set the **Target scope** property to `Enabled`.

9 **Target mouse** allows you to disable or enable mouse support on the target PC. From the **Target mouse** list, select `None`, `PS2`, `RS232 COM1`, or `RS232 COM2`.

Changing Environment Properties with a Command-Line Interface

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

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

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

1 In the MATLAB window, type

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

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

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

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

2 In the MATLAB window, type

```
updatexpcenv
```

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

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

xPC Target Demos

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

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

To Locate or Edit a Demo Script

1 In the MATLAB Command Window, type

```
which scfreerundemo
```

MATLAB displays the location of the M-file.

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

2 Type

```
edit scfreerundemo
```

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

6

Using the Target PC Command-Line Interface

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

Target PC Command-Line Interface

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

For a complete list of target PC commands, refer to Chapter 13, "Target PC Commands."

Using Target Application Methods on the Target PC

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

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

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

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

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

   ```
   start
   ```

3. To stop the application, type

   ```
   stop
   ```

Once the command window is active, you do not have to reactivate it before typing the next command.

Manipulating Target Object Properties from the Target PC

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

These commands create a temporary difference between the behavior of the target application and the properties of the target object. The next time you access the target object, the properties are updated from the target PC.

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

   The target PC activates the command window.

2 Type a target command. For example, to change the frequency of the signal generator (parameter 1) in the model `xpcosc`, type

   ```
   setpar 1=30
   ```

   The command window displays a message to indicate that the new parameter has registered.

   ```
   System: p[1] is set to 30.00000
   ```

3 Check the value of parameter 1. For example, type

   ```
   p1
   ```

   The command window displays a message to indicate that the new parameter has registered.

   ```
   System: p[1] is set to 30.00000
   ```

4 Check the value of signal 0. For example, type

   ```
   s0
   ```

The command window displays a message to indicate that the new parameter has registered.

```
System: S0 has value 5.1851
```

5 Change the stop time. For example, to set the stop time to 1000, type

```
stoptime = 1000
```

The parameter changes are made to the target application but not to the target object. When you type any xPC Target command in the MATLAB Command Window, the target PC returns the current properties of the target object.

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

To see the correlation between a parameter or signal index and its block, you can look at the `model_name_pt.c` or `model_name_bio.c` of the generated code for your target application.

Manipulating Scope Objects from the Target PC

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

These commands create a temporary difference between the behavior of the target application and scope object. The next time you access the scope object, the data is updated from the target PC.

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

   The target PC activates the command window.

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

```
addscope 2
```

xPC Target adds another scope monitor to the target PC screen. The command window displays a message to indicate that the new scope has registered.

```
Scope: 2, created, type is target 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 as many signals as necessary to the scope.

4 Type a scope command. For example, to start the scope `2`, type

```
startscope 2
```

The target scope `2` starts and displays the signals you added in the previous step.

Note If you add a target scope from the target PC, you need to start that scope manually. If a target scope is in the model, starting the target application starts that scope automatically.

Manipulating Scope Object Properties from the Target PC

This section shows some of the common tasks that you use with target objects and their properties.

These commands create a temporary difference between the behavior of the target application and the properties of the target object. The next time you access the target object, the properties are updated from the target PC.

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

   The target PC activates the command window.

2 Type a scope property command. For example, to change the number of samples (1000) to acquire in scope `2` of the model `xpcosc`, type

   ```
   numsamples 2=1000
   ```

3 Type a scope property command. For example, to change the scope mode (numerical) of scope `2` of the model `xpcosc`, type

   ```
   scopemode 2=numerical
   ```

   The target scope `2` display changes to a numerical one.

Aliasing with Variable Commands on the Target PC

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

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

1 On the target PC keyboard, type a variable command. For example, if you have a parameter that controls a motor, you could create the variables `on` and `off` by typing

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

   The target PC command window is activated when you start to type, and a command line opens.

2 Type the variable name to run that command sequence. For example, to turn the motor on, type

```
on
```

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

7

Working with Target PC Files and File Systems

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

Introduction

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

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

The largest single file that you can create is 4 GB.

Note that writing data files to 3.5 inch disk drives is considerably slower than writing to hard drives.

You can access signal data files, or any target PC system file, in one of the following ways:

- If you are running the target PC as a stand-alone system, you can access that file by rebooting the target PC under an operating system such as DOS and accessing the file through the operating system utilities.
- If you are running the target PC in conjunction with a host PC, you can access the target PC file from the host PC by representing that file as an `xpctarget.ftp` object. Through the MATLAB interface, use `xpctarget.ftp` methods on that FTP object. The `xpctarget.ftp` object methods are file transfer operations such as `get` and `put`.
- If you are running the target PC in conjunction with a host PC, you can access the target PC file from the host PC by representing the target PC file system as an `xpctarget.fs` object. Through the MATLAB interface, use the `xpctarget.fs` methods on the file system and perform file system-like methods such as `fopen` and `fread` on the signal data file. These methods work like the MATLAB file I/O methods. The `xpctarget.fs` methods also include file system utilities that allow you to collect target PC file system information for the disk and disk buffers.

This chapter describes procedures on how to use the `xpctarget.ftp` and `xpctarget.fs` methods for common operations. See Chapter 14, “Function Reference,” for a reference of the methods for these objects.

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

FTP and File System Objects

xPC Target uses two objects, `xpctarget.ftp` and `xpctarget.fs` (file system), to work with files on a target PC. You use the `xpctarget.ftp` object to perform file transfer operations between the host and target PC. You use the `xpctarget.fs` object to access the target PC file system. For example, you can use an `xpctarget.fs` object to open, read, and close a signal data file created by an xPC Target scope of type `file`.

Note This feature provides FTP-like commands, such as `get` and `put`. However, it is not a standard FTP implementation. For example, xPC Target 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`.

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

To create an `xpctarget.fs` object, 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`. xPC Target creates the `xpctarget.fsbase` object when you create either an `xpctarget.ftp` or `xpctarget.fs` object.

Using xpctarget.ftp Objects

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

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

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

This section includes the following topics:

xPC Target also provides methods that allow you to perform file system-type manipulations, such as opening and reading files. For a complete list of these methods, see “Using xpctarget.fs Objects” on page 7-9.

Accessing Files on a Specific Target PC

You can access specific target PC files from the host PC for the `xpctarget.ftp` object.

Use the `xpctarget.ftp` creator function. If your system has multiple targets, you can access specific target PC files from the host PC for the `xpctarget.ftp` object.

For example, to list the name of the current directory of a target PC through a TCP/IP connection,

1 In the MATLAB window, type a command like the following to assign the `xpctarget.ftp` object to a variable.

```
f=xpctarget.ftp('TCPIP','192.168.0.1','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','192.168.0.1','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 PC, you can use the `xpctarget.ftp` constructor without arguments.

1 In the MATLAB window, type a command like the following to assign the `xpctarget.ftp` object to a variable.

```
f=xpctarget.ftp;
```

xPC Target assigns the `f` variable to the default target PC.

Listing the Contents of the Target PC Directory

You can list the contents of the target PC directory by using xPC Target methods on the host PC for the `xpctarget.ftp` object. Use the method syntax to run an `xpctarget.ftp` object method:

```
method_name(ftp_object)
```

Note You must use the `dir(f)` syntax to list the contents of the directory. To get the results in an M-by-1 structure, use a syntax like `y=dir(f)`. See the `dir` method reference for further details.

For example, to list the contents of the C:\ drive,

1 In the MATLAB window, type the following to assign the `xpctarget.ftp` object to a variable:

   ```
   f=xpctarget.ftp;
   ```

2 Type

   ```
   f.pwd
   ```

   This gets the current directory. You get a result like the following:

   ```
   ans =
   C:\
   ```

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

   ```
   dir(f)
   ```

Retrieving a File from the Target PC to the Host PC

You can retrieve a copy of a data file from the target PC by using xPC Target methods on the host PC for the `xpctarget.ftp` object.

Use the method syntax to run an `xpctarget.ftp` object method. The syntax `method_name(ftp_object, argument_list)` can be replaced with

```
ftp_object.method_name(argument_list)
```

For example, to retrieve a file named `data.dat` from the target PC C:\ drive (default),

1 If you have not already done so, in the MATLAB window, type the following to assign the `xpctarget.ftp` object to a variable.

   ```
   f=xpctarget.ftp;
   ```

2 Type

```
f.get('data.dat');
```

This retrieves the file and saves that file to the variable `data`. This content is in the xPC Target file format.

Copying a File from the Host PC to the Target PC

You can place a copy of a file from the host PC by using xPC Target methods on the host PC for the `xpctarget.ftp` object.

Use the method syntax to run an `xpctarget.ftp` object method. The syntax `method_name(ftp_object, argument_list)` can be replaced with

```
ftp_object.method_name(argument_list)
```

For example, to copy a file named `data2.dat` from the host PC to the target PC C:\ drive (default),

1 If you have not already done so, in the MATLAB window, type the following to assign the `xpctarget.ftp` object to a variable.

```
f=xpctarget.ftp;
```

2 Type the following to save that file to the variable `data`.

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

Using xpctarget.fs Objects

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

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

Useful global utility:

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

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

This section includes the following topics:

- "Accessing File Systems from a Specific Target PC" on page 7-10
- "Retrieving the Contents of a File from the Target PC to the Host PC" on page 7-11
- "Removing a File from the Target PC" on page 7-13
- "Getting a List of Open Files on the Target PC" on page 7-14

- "Getting Information about a File on the Target PC" on page 7-15
- "Getting Information about a Disk on the Target PC" on page 7-15

xPC Target also provides methods that allow you to perform file transfer operations, such as putting files on and getting files from a target PC. For a description of these methods, see "Using xpctarget.ftp Objects" on page 7-5.

Accessing File Systems from a Specific Target PC

You can access specific target PC files from the host PC for the `xpctarget.fs` object.

Use the `xpctarget.fs` creator function. If your system has multiple targets, you can access specific target PC files from the host PC for the `xpctarget.fs` object.

For example, to list the name of the current directory of a target PC 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','192.168.0.1','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','192.168.0.1','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 PC, 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;
```

xPC Target assigns the `fsys` variable to the default target PC.

2 Type

```
fsys.dir;
```

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

You can retrieve the contents of a data file from the target PC by using xPC Target methods on the host PC for the `xpctarget.fs` object.

Use the method syntax to run an `xpctarget.fs` object method. The syntax `method_name(fs_object, argument_list)` can be replaced with

```
fs_object.method_name(argument_list)
```

For example, to retrieve the contents of a file named `data.dat` from the target PC C:\ drive (default),

1 If you have not already done so, in the MATLAB window, type the following to assign the `xpctarget.fs` object to a variable.

```
fsys=xpctarget.fs;
```

2 Type

```
h=fsys.fopen('data.dat');
```

or

```
h=fopen(fsys,'data.dat');
```

This opens the file `data.dat` for reading and assigns the file identifier to `h`.

3 Type

```
data2=fsys.fread(h);
```

or

```
data2=fread(fsys,h);
```

This reads the file `data.dat` and stores the contents of the file to `data2`. This content is in the xPC Target file format.

4 Type

```
fsys.fclose(h);
```

This closes the file `data.dat`.

Before you can view or plot the contents of this file, you must convert the contents. See “Converting xPC Target File Format Content to Bytes” on page 7-12.

Converting xPC Target File Format Content to Bytes

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

This section assumes that you have a variable, `data2`, that contains data in the xPC Target file format (see “Retrieving the Contents of a File from the Target PC to the Host PC” on page 7-11):

1 In the MATLAB window, change directory to the directory that contains the xPC Target format file.

2 Type

```
new_data2=readxpcfile(data2);
```

The `readxpcfile` script converts the format of `data2` from the xPC Target file format to an array of bytes. It also creates a structure for that file in `new_data2`, of which one of the elements is an array of doubles, `data`. The `data` member is also appended with a time stamp vector. All data is returned as doubles, which represent the real-world 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 xPC Target in StandAlone mode, you can extract the data from the data file if you know the number of signals in the scope. If you know this number, you can extract the data. Note the following:

- Ignore the first 512 bytes of the file. This is file header information.
- After the first 512 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 PC

You can remove a file from the target PC by using xPC Target methods on the host PC 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 PC C:\ drive (default),

1. If you have not already done so, in the MATLAB window, type the following to assign the `xpctarget.fs` object to a variable.

   ```
   fsys=xpctarget.fs;
   ```

2. Type the following to remove the specified file from the target PC.

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

   or

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

Getting a List of Open Files on the Target PC

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

Use the method syntax to run an `xpctarget.fs` object method. The syntax `method_name(fs_object, argument_list)` can be replaced with

```
fs_object.method_name(argument_list)
```

For example, to get a list of open files for the file system object `fsys`,

1. If you have not already done so, in the MATLAB window, type the following to assign the `xpctarget.fs` object to a variable.

   ```
   fsys=xpctarget.fs;
   ```

2. Type

   ```
   fsys.filetable
   ```

   If the file system has open files, a list like the following is displayed:

   ```
   ans =
   Index    Handle  Flags     FilePos  Name
   ---------------------------------------
       0  00060000  R__          8512  C:\DATA.DAT
       1  00080001  R__             0  C:\DATA1.DAT
       2  000A0002  R__          8512  C:\DATA2.DAT
       3  000C0003  R__          8512  C:\DATA3.DAT
       4  001E0001  R__             0  C:\DATA4.DAT
   ```

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

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

Use the method syntax to run an `xpctarget.fs` object method. The syntax `method_name(fs_object, argument_list)` can be replaced with

```
fs_object.method_name(argument_list)
```

For example, to display the information for the file identifier `fid1`,

1 If you have not already done so, in the MATLAB window, type the following to assign the `xpctarget.fs` object to a variable.

```
fsys=xpctarget.fs;
```

2 Type

```
fid1=fsys.fopen('data.dat');
```

This opens the file `data.dat` for reading and assigns the file identifier to `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 PC

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

Use the method syntax to run an `xpctarget.fs` object method. The syntax `method_name(fs_object, argument_list)` can be replaced with

```
fs_object.method_name(argument_list)
```

For example, to display the disk information for the C:\ drive,

1 If you have not already done so, in the MATLAB window, type the following to assign the `xpctarget.fs` object to a variable.

```
fsys=xpctarget.fs;
```

2 Type

```
fsys.diskinfo('C:\');
```

This returns disk information like the following for the C:\ drive file system.

```
ans =
                  Label: 'SYSTEM '
            DriveLetter: 'C'
               Reserved: ''
           SerialNumber: 1.0294e+009
    FirstPhysicalSector: 63
                FATType: 32
               FATCount: 2
          MaxDirEntries: 0
         BytesPerSector: 512
      SectorsPerCluster: 4
          TotalClusters: 2040293
            BadClusters: 0
           FreeClusters: 1007937
                  Files: 19968
             FileChains: 22480
             FreeChains: 1300
       LargestFreeChain: 64349
```

Graphical User Interfaces

You can run and test your target application using the MATLAB command-line interface or the Simulink block diagram for your application. You can also use special blocks provided with xPC Target to interface signals and parameters from a target application to a custom GUI application. This chapter includes the following sections:

xPC Target Interface Blocks to Simulink Models

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

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

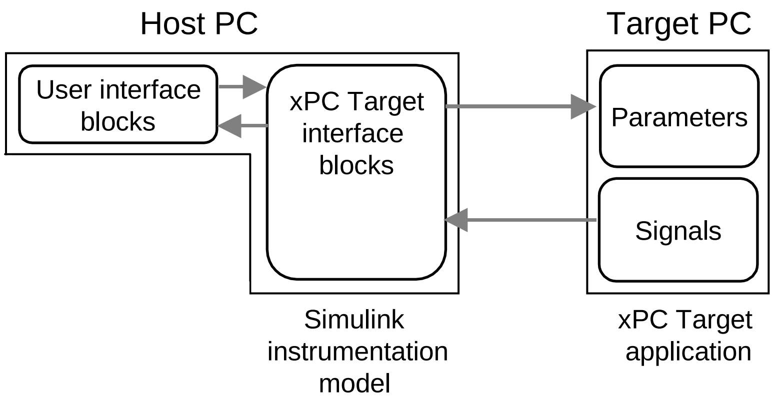
Simulink User Interface Model

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

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

Virtual Reality Toolbox — The Virtual Reality Toolbox enables you to display a Simulink user interface model in 3-D. It provides Simulink blocks that communicate with xPC Target interface blocks. These blocks then communicate 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

xPC Target provides Simulink interface blocks to connect graphical interface elements to your target application. The steps for creating your own custom user interface are listed below:

1. In the Simulink target application model, decide which block parameters and block signals you want to have access to through graphical interface control devices and graphical interface display devices.

2. Tag all block parameters in the Simulink model that you want to be connected to a control device. See "Marking Block Parameters" on page 8-8.

3. Tag all signals in Simulink model that you want to be connected to a display device. See "Marking Block Signals" on page 8-10.

4. In MATLAB, run the function `xpcsliface('model_name')` to create the user interface template model. This function generates a new Simulink model containing only the xPC Target interface blocks (To xPC Target and From xPC Target) defined by the tagged block parameters and block signals in the target application model.

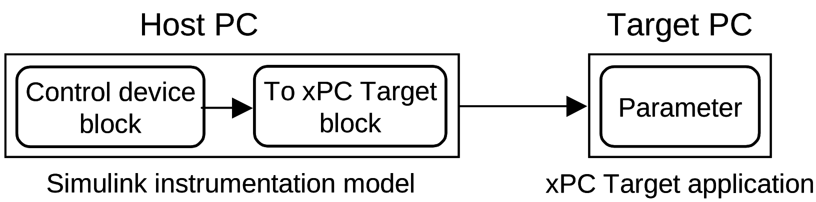
5. To the user interface template model, add Simulink interface blocks from add-on products (Virtual Reality Toolbox, Altia Design).
   - You can connect Altia blocks to the xPC Target To PC Target interface blocks. To xPC Target blocks on the left should be connected to control devices.
   - You can connect Altia and Virtual Reality Toolbox 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.

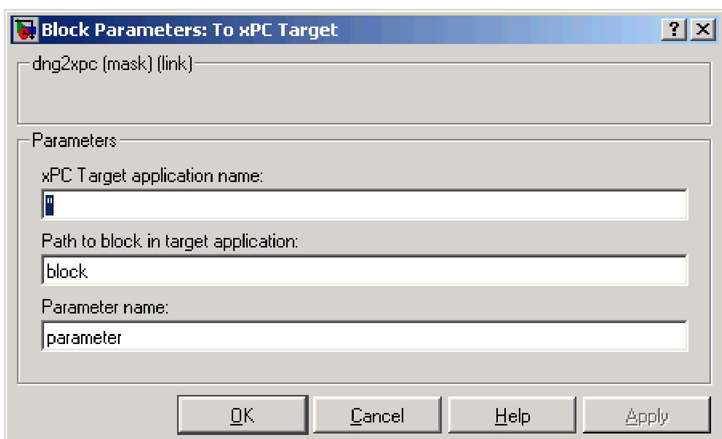
6. Start both the xPC target application and the Simulink user interface model that represents the xPC Target application.

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 an M-file S-function. The block is optimized so that it only changes a parameter on the target application when the input value differs from the value that existed at the last time step. This block uses the parameter downloading feature of the xPC command-line interface. This block is available from the **xpclib/Misc** block sublibrary. See "To xPC Target" on page 34-12 in the xPC Target I/O reference documentation for further configuration details.



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

Block Parameters

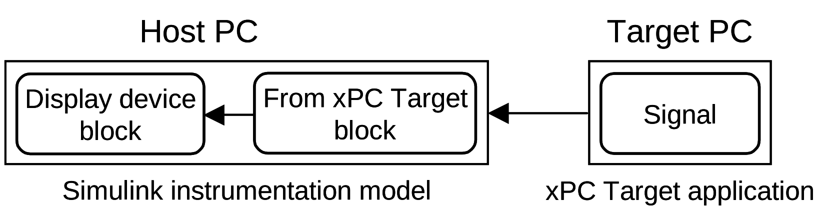
xPC Target application name — The function `xpcsliface` automatically enters a name entry for this parameter. It is the same name as the Simulink model that xPC Target uses to build the target application.

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

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

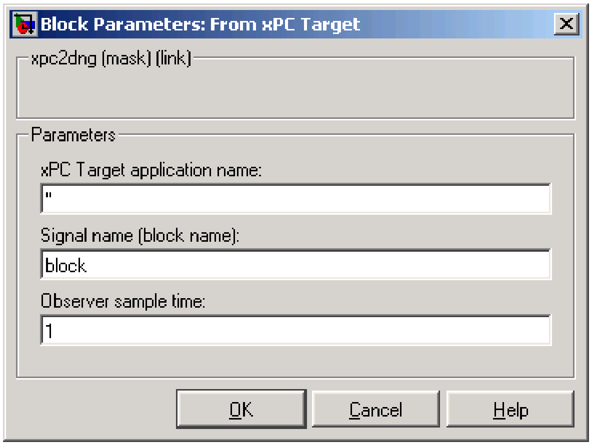
From xPC Target Block

This block behaves like a source and its output is usually connected to the input of a display device.



Because only one numerical value per signal is uploaded during a time step, the number of samples of a scope object is set to 1. The block uses the signal tracing capability of the xPC Target command-line interface and is implemented as an M-file S-function. This block is available from the

xpclib/Misc sublibrary. See “From xPC Target” on page 34-10 in the xPC Target I/O reference documentation for further configuration details.



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

Block Parameters

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

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

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

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 8-8 and “Marking Block Signals” on page 8-10 for descriptions of how to mark block properties and block signals.

Marking Block Parameters

Tagging parameters in your Simulink model allows the function `xpcsliface` to create To xPC Target interface blocks. These interface blocks contain the parameters you connect to control devices in your user interface model.

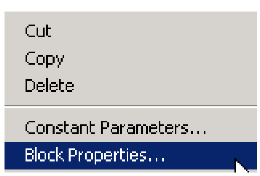
After you create a Simulink model, you can mark the block parameters. This procedure uses the model `xpctank.mdl` as an example.

1 Open a Simulink model. For example, in the MATLAB Command Window, type

```
xpctank
```

2 Point to a Simulink block, and then right-click.

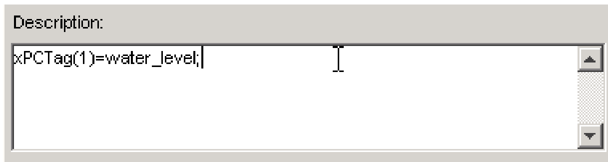
3 From the menu, click **Block Properties**. Do not click **Constant Parameters**.



A Block properties dialog box opens.

4 In the **Description** box, delete the existing tag and enter a tag to the parameters for this block.

For example, the SetPoint block is a constant with a single parameter that selects the level of water in the tank. Enter the tag shown below.



The tag has the following format syntax

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

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

```
xPCTag=label;
```

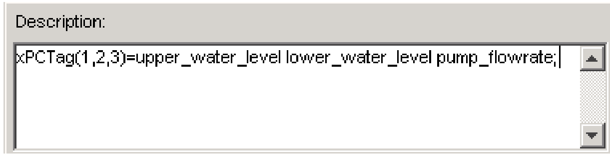
`index_n` – Index of a block parameter. Begin numbering parameters with an index of `1`.

`label_n` – Name for a block parameter that will be connected to a To xPC Target block in the user interface model. Separate the labels with a space, not a comma.

`label_1...label_n` must consist of the same identifiers as those used by C/C++ to name functions, variables, and so forth. Do not use names like `-foo`.

5 Repeat steps 1 through 3 for the remaining parameters you want to tag.

For example, for the Controller block, enter the tag



For the PumpSwitch and ValveSwitch blocks, enter the following tags respectively:

```
xPCTag(2)=pump_switch;
```

```
xPCTag(1)=drain_valve;
```

To create the To xPC blocks in an user interface model for a block with four properties, use the following syntax:

```
xPCTag(1,2,3,4)=label_1 label_2 label_3 label_4;
```

To create the To xPC blocks for the second and fourth properties in a block with at least four properties, use the following syntax:

```
xPCTag(2,4)=label_1 label_2;
```

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

```
xpc_tank1
```

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 8-10 and "Creating a Custom Graphical Interface" on page 8-3.

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 `xpc_tank1.mdl` (or `xpctank.mdl`) as an example. See "Creating a Target Application Model" on page 8-8.

Note that you cannot select signals on the output ports of any virtual blocks such as Subsystem and Mux blocks. Also, you cannot select signals on any function-call, triggered signal output ports.

1 Open a Simulink model. For example, in the MATLAB Command Window, type

   `xpc_tank` or `xpc_tank1`

2 Point to a Simulink signal line, and then right-click.

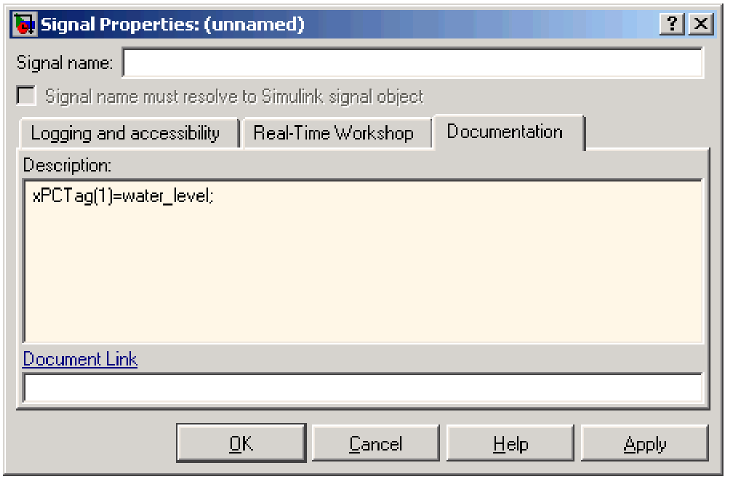
3 From the menu, click **Signal Properties**.

   A Signal Properties dialog box opens.

4 Select the **Documentation** tab.

5 In the **Description** box, enter a tag to the signals for this line.

   For example, the block labeled TankLevel is an integrator with a single signal that indicates the level of water in the tank. Replace the existing tag with the tag shown below.



The tag has the following format syntax:

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

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

```
XPCTag=label:
```

- `index_n` — Index of a signal within a vector signal line. Begin numbering signals with an index of `1`.
- `label_n` — Name for a signal that will be connected to a From xPC Target block in the user interface model. Separate the labels with a space, not a comma.

`label_1...label_n` must consist of the same identifiers as those used by C/C++ to name functions, variables, and so forth. Do not use names like `-foo`.

To create the From xPC blocks in an user interface model for a signal line with four signals (port dimension of 4), use the following syntax:

```
xPCTag(1,2,3,4)=label_1 label_2 label_3 label_4;
```

To create the From xPC blocks for the second and fourth signals in a signal line with at least four signals, use the following syntax:

```
xPCTag(2,4)=label_1 label_2;
```

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

```
xpc_tank1
```

You next task is to mark block parameters if you have not already done so. See “Marking Block Parameters” on page 8-8. If you have already marked block signals, return to “Creating a Custom Graphical Interface” on page 8-3 for additional guidance on creating a user interface template model.

9

xPC Target Web Browser Interface

xPC Target has a Web server that allows you to interact with your target application through a Web browser. You can access the Web browser with either a TCP/IP or serial (RS-232) connection. This chapter includes the following section:

Web Browser Interface

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

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

This section includes the following topics:

Connecting the Web Interface Through TCP/IP

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

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

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

1 In the MATLAB window, type

```
xpcwwwenable
```

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

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

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

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

Connecting the Web Interface Through RS-232

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

1 In the MATLAB window, type

`xpcwwwenable` or `close(xpc)`

MATLAB is disconnected from the target PC, leaving the target PC ready to connect to another client. The TCP/IP stack of the xPC Target kernel supports only one simultaneous connection. If you do not use this command, the TCP/IP to RS-232 gateway might not be able to connect to the target PC.

2 Open a DOS command window, and enter the command to start the TCP/IP to RS-232 gateway. For example, if the target PC is connected to COM1 and you would like to use the TCP/IP port 22222, type the following:

```
c:\<MATLAB root>\toolbox\rtw\targets\xpc\xpc\bin\xpctcp2ser -v
-t 22222 -c 1
```

For a description of the xpctcp2ser command, see “Syntax for the xpctcp2ser Command” on page 9-5.

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, then `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
```

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

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

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

You must close MATLAB and then restart it.

Syntax for the xpctcp2ser Command

The `xpctcp2ser` command starts the TCP/IP to RS-232 gateway. The syntax for this command is

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

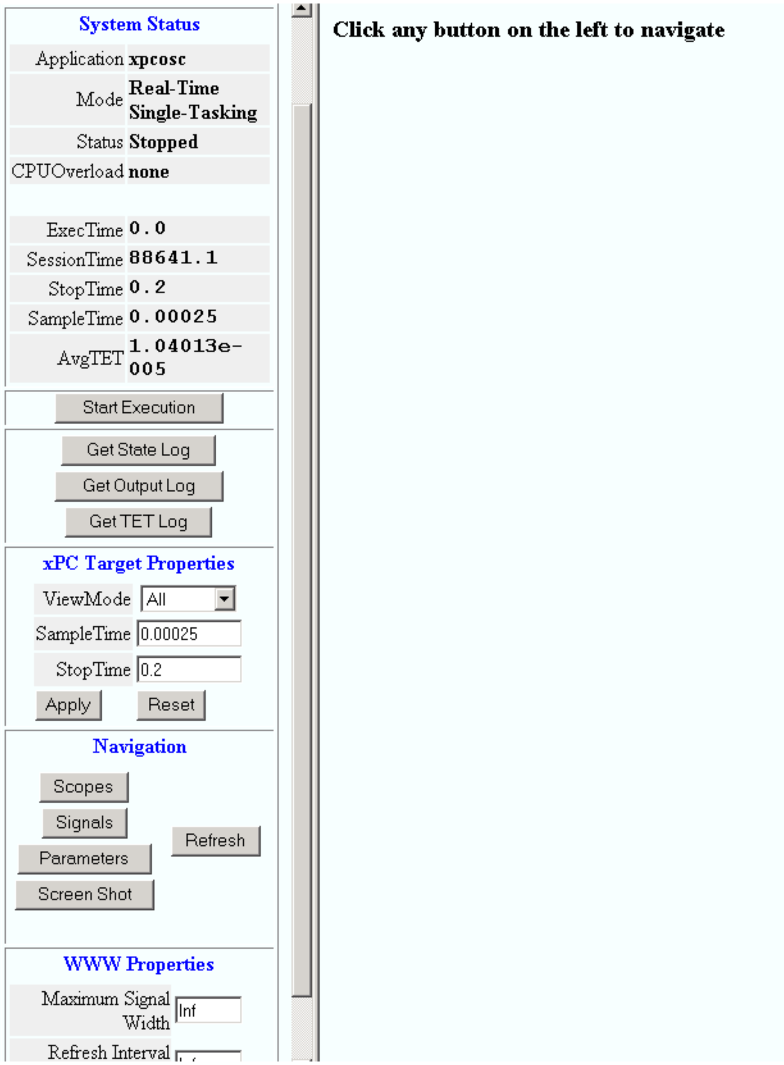
The options are described in the following table.

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 PC that is connected to the target PC with a serial cable can connect to the selected port. For example, if you start the gateway on your host PC, with the default ports, you can type in the Web browser `http://localhost:2222`. However, if you try to connect to `http://Domainname.com:22222`, you will probably get a connection error.

Command-Line Option	Description
`-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 PC, you can use the **Main** pane to control the target application:

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

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

System Status

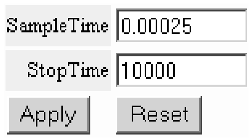
Application	**xpcosc**
Mode	**Real-Time Single-Tasking**
Status	**Stopped**
CPUOverload	**none**
ExecTime	**0.0**
SessionTime	**97769**
StopTime	**10000**
SampleTime	**0.00025**
AvgTET	**-nan**

2 Click the **Start Execution** button.

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

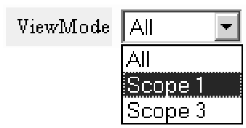
3 Update the execution time and average task execution time (TET). Click the **Refresh** button. To stop the target application, click the **Stop Execution** button.

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



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

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



Note The **ViewMode** control is visible in the **xPC Target Properties** pane only if you add two or more scopes to the target PC.

Changing WWW Properties

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

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

Signals with a width greater than the value you enter are not displayed on the **Signals** pane.

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

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

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

This can also happen when you are trying to update a parameter or property at the same time that the pane is automatically refreshing.

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

Viewing Signals with a Web Browser

The **Signals** pane is a list of the signals in your model.

After you connect a Web browser to the target PC you can use the **Signals** pane to view signal data:

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

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

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

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

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

   The **Screen Shot** pane is loaded and a copy of the current target PC screen is displayed. The screen shot uses the portable network graphics (PNG) file format.

Viewing Parameters with a Web Browser

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

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

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

   The **Parameter List** pane is loaded into the right frame.

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

   If the parameter is a vector or matrix, press the **Edit** button to view the vector or matrix (in the correct shape). You can edit the parameter in this pane.

2. In the **Value** box, enter a new parameter value, and then click the **Apply** button.

Changing Access Levels to the Web Browser

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

1. In the Simulink window, click **Configuration Parameters**.

   The Configuration Parameters dialog box for the model is displayed.

2. Click the **Real-Time Workshop** node.

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

3 In the **Target selection** section, access levels are set in the **RTW system target file** box. For example, to set the access level to `1`, enter

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

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

4 Click **OK**.

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

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

Level 0 — Full access to all panes and functions.

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

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

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

Level 4 — Cannot edit existing scopes on the **Scopes** pane. Cannot add or remove signals on the **Scopes** pane. Cannot view the **Signals** pane and the **Parameters** pane, and cannot get scope data.

10

Interrupts Versus Polling

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

Polling Mode

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

xPC Target Kernel Polling Mode

Polling mode for the xPC Target real-time kernel is designed to execute target applications at sample times close to the limit of the hardware (CPU). Using polling mode with high-speed and low-latency I/O boards and drivers allows you to achieve smaller sample times for applications that you cannot achieve using the interrupt mode of xPC Target.

Polling mode has two main applications:

- **Control applications** — Control applications of average model size and I/O complexity that are executed at very small sample times (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)

Interrupt Mode

Interrupt mode is the default real-time execution mode for the xPC Target kernel. This mode provides the greatest flexibility and is the mode you should choose for any application that executes at the given base sample time without overloading the CPU.

The scheduler ensures real-time single-tasking and multitasking execution of single-rate or multirate systems, including asynchronous events (interrupts).

Additionally, background tasks like host-target communication or updating the target screen run in parallel with sample-time-based model tasks. This allows you to interact with the target system while the target application is executing in real time at high sample rates. This is made possible by an interrupt-driven real-time scheduler that is responsible for executing the various tasks according to their priority. The base sample time task can interrupt any other task (larger sample time tasks or background tasks) and execution of the interrupted tasks resumes as soon as the base sample time task completes operation. This gives a quasi parallel execution scheme with consideration to the priorities of the tasks.

Latencies Introduced by Interrupt Mode

Compared to other modes, interrupt mode has more advantages. The exception is the disadvantage of introducing a constant overhead, or latency, that reduces the minimal possible base sample time to a constant number. The overhead is the sum of various factors related to the interrupt-driven execution scheme and can be referred to as overall interrupt latency. The overall latency consists of the following parts, assuming that the currently executing task is not executing a critical section and has therefore not disabled any interrupt sources:

- **Interrupt controller latency** — In a PC-compatible system the interrupt controller is not part of the x86-compatible CPU but part of the CPU chip set. The controller is accessed over the I/O-port address space, which introduces a read or write latency of about 1 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 to properly handle the interrupt controller.
- **CPU hardware latency** — Modern CPUs try to predict the next couple of instructions, including branches, by the use of instruction pipelines. If an interrupt occurs, the prediction fails and the pipeline has to be fully reloaded. This process introduces an additional latency. Additionally, because of interrupts, cache misses will occur.

- **Interrupt handler entry and exit latency** — Because an interrupt can stop the currently executing task at any instruction and the interrupted task has to resume proper execution when the interrupting task completes execution, its state has to be saved and restored accordingly. This includes saving CPU data and address registers, including the stack pointer. In the case that the interrupted task executed floating-point unit (FPU) operations, the FPU stack has to be saved as well (108 bytes on a Pentium CPU). This introduces additionally latency.

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

The xPC Target real-time kernel in interrupt mode is close to optimal for executing code on a PC-compatible system. However, interrupt mode introduces an overall latency of about 8 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, a certain amount of headroom (at least 5%) is necessary, which can cause additional cache misses and therefore nonoptimal execution speed.

Polling Mode

Polling mode for the xPC Target real-time kernel does not have the 8 µs of latency that interrupt mode does. This is because the kernel does not allow interrupts at all, so the CPU can use this extra time for executing model code.

Polling mode is sometimes seen as a “primitive” or “brute force” real-time execution scheme. Nevertheless, when a real-time application executes at a given base sample time in interrupt mode and overloads the CPU, switching to polling mode is often the only alternative to get the application to execute at the required sample time.

Polling means that the kernel waits in an empty while loop until the time at which the next model step has to be executed is reached. Then the next model step is executed. At least a counter implemented in hardware has to be accessible by the kernel in order to get a base reference for when the next model step execution has to commence. The kernel polls this hardware counter. If this hardware counter must be outside the CPU, e.g., in the chip set or even on an ISA or PCI board, the counter value can only be retrieved by an I/O or memory access cycle that again introduces latency. This latency usually eats up the freed-up time of polling mode. Fortunately, since the introduction of the Pentium CPU family from Intel, the CPU is equipped with a 64 bit counter on the CPU substrate itself, which commences counting at power-up time and counts up driven by the actual clock rate of the CPU. Even a highly clocked CPU is not likely to lead to an overflow of a 64 bit counter (2^64 * 1e-9 (1 GHz CPU) = 584 years). The Pentium counter comes with the following features:

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

The polling execution scheme does not depend on any interrupt source to notify the code to continue calculating the next model step. While this frees the CPU, it means that any code that is part of the exclusively running polling loop is executed in real time, even components, which have so far been executed in background tasks. Because these background tasks are usually non-real-time tasks and can use a lot of CPU time, do not execute them. This is the main disadvantage of polling mode. To be efficient, only the target application's relevant parts should be executed. In the case of xPC Target, this is the code that represents the Simulink model itself.

Therefore, host-target communication and target display updating are disabled. Because polling mode reduces the features of xPC Target to a minimum, you should choose it only as the last possible alternative to reach the required base sample time for a given model. Therefore, ensure the following before you consider polling mode:

- **The model is optimal concerning execution speed** — First, you should run the model through the Simulink profiler to find any possible speed optimizations using alternative blocks. If the model contains continuous states, the discretization of these states will reduce model complexity significantly, because a costly fixed-step integration algorithm can be avoided. If continuous states cannot be discretized, you should use the integration algorithm with the lowest order that still produces correct numerical results.
- **Use the fastest available computer hardware** — Ensure that the CPU with the highest clock rate available is used for a given PC form factor. For the desktop form factor, this would mean a clock rate above 1 GHz; for a mobile application, e.g., using the PC/104 form factor, this would mean a clock rate above 400 MHz. Most of the time, you should use a desktop PC, because the highest clocked CPUs are available for this form factor only. Executing `xpcbench` at the MATLAB prompt gives an understanding about the best performing CPUs for xPC Target applications.
- **Use the lowest latency I/O hardware and drivers available** — Many xPC Target applications communicate with hardware through I/O hardware over either an ISA or PCI bus. Because each register access to such I/O hardware introduces a comparably high latency time (~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 increasing performance.

Setting the Polling Mode

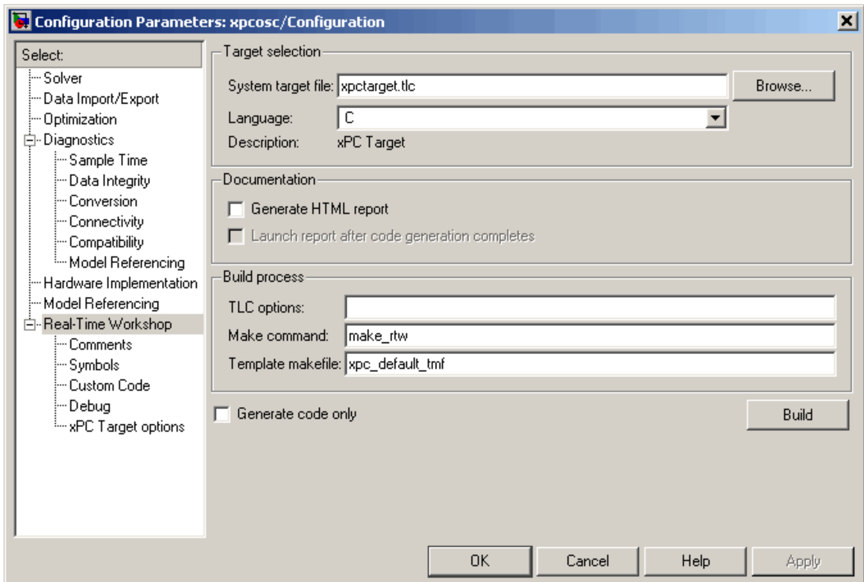
Polling mode is an alternative to the default interrupt mode of the real-time kernel. This means that the kernel on the bootable 3.5 inch disk created by the `xpcexplr` GUI allows running the target application in both modes without the necessity to use another boot disk.

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

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

The Configuration Parameters dialog box opens.

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



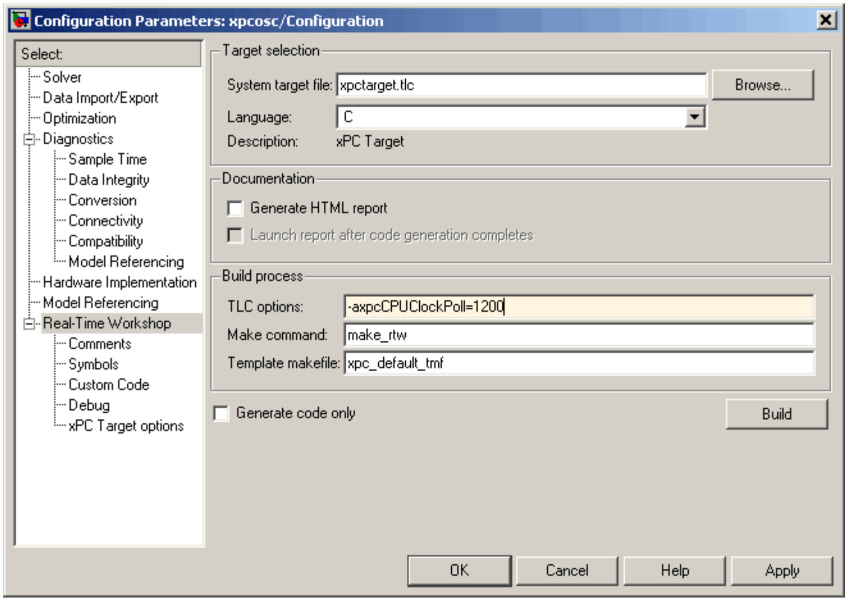
3 In the **TLC options** edit field, specify the option

`-axpcCPUClockPoll=`*`CPUClockRateMHz`*

The assignment of the clock rate of the target PC's CPU is necessary because the Pentium's on-chip counter used for polling mode counts up with the CPU clock rate. If the clock rate is provided, the kernel can convert clock ticks to seconds and vice versa. If an incorrect clock rate is provided, the target application executes at an incorrect base sample time. You can find out about the CPU clock rate of the target PC by rebooting the target PC and checking the screen output during BIOS execution time. The BIOS usually displays the CPU clock rate in MHz right after the target PC has been powered up.

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

```
-axpcCPUClockPoll=1200
```



If you want to execute the target application in interrupt mode again, either remove the option or assign a CPU clock rate of 0 to the option:

```
-axpcCPUClockPoll=0
```

If you make a change to the **TLC options** field, you need to rebuild the target application for the change to take effect. Building the target application, downloading it, and preparing it for a run then work exactly the same way as they did with default interrupt mode.

After the download of the target application has succeeded, the target screen displays the mode, and if polling mode is activated, it additionally displays the defined CPU clock rate in MHz. This allows checking for the correct setting.

Restrictions Introduced by Polling Mode

As explained above, polling mode executes the Simulink-based target application in real time exclusively. While the target application is executing in polling mode, the background tasks, mainly the ones for host-target communication and target screen updating, are inactive. This is because all interrupts of the target PC are fully disabled during the execution of the target application. On one hand this ensures the highest polling performance; on the other hand, as a consequence the background tasks are not serviced.

Here is a list of all relevant restrictions of polling mode, which are otherwise available in the default interrupt mode.

Host-Target Communication Is Not Available During the Execution of the Target Application

If the target application execution is started in polling mode, e.g., with

```
start(tg)
```

host-target communication is disabled throughout the entire run, or in other words until the stop time is reached. Each attempt to issue a command like

```
tg
```

leads to a communication-related error message. Even the `start(tg)` command to start polling mode execution returns such an error message, because the host side does not receive the acknowledgment from the target before timing out. The error message when executing `start(tg)` is not avoidable. Subsequently, during the entire run, it is best not to issue any target-related commands on the host, in order to avoid displaying the same error message over and over again.

As a consequence, it is not possible to issue a `stop(tg)` command to stop the target application execution from the host side. The target application has to reach its set stop time for polling mode to be exited. You can use

```
tg.stoptime=x
```

before starting the execution, but once started the application executes until the stop time is reached.

Nevertheless, there is a way to stop the execution interactively before reaching the target application stop time. See “Controlling the Target Application” on page 10-12.

If the target application execution finally reaches the stop time and polling mode execution is stopped, host-target communication will begin functioning again. However, the host-target communication link might be in a bad state. If you still get communication error messages after polling mode execution stops, type the command

```
xpctargetping
```

to reset the host-target communication link.

After the communication link is working again, type

```
tg
```

to resync the target object on the host side with the most current status of the target application.

Target Screen Does Not Update During the Execution of the Target Application

As with the restriction mentioned above, target screen updating is disabled during the entire execution of the target application. Using the kernel with the **Enable target scope** option enabled (see `xpcexplr` GUI) does not work. You should therefore use the kernel with the **Enable target scope** property disabled (text output only). The kernel enabled with text mode actually provides more information when running in polling mode.

Session Time Does Not Advance During the Execution of the Target Application

Because all interrupts are disabled during a run, the session time does not advance. The session time right before and after the run is therefore the same. This is a minor restriction and should not pose a problem.

The Only Rapid-Prototyping Feature Available Is Data Logging

Because host-target communication and target screen updating are disabled during the entire run, most of the common rapid-prototyping features of xPC Target are not available in polling mode. These are

- Parameter tuning — Neither through the command-line interface nor through External mode

- Signal tracing through scope objects — Neither through scope objects of type `host` (**xPC Target Explorer** or scripts) or type `target` (scopes on the target screen if property **Enable target scope** is enabled)
- Signal monitoring — You cannot run a GUI interface on the host PC using an environment that depends on communication between the host and target computers.
- Applications using the xPC Target API
- The Internet browser interface
- Other utilities like `xpctargetspy`

The only rapid-prototyping feature available is signal logging, because the acquisition of signal data runs independently from the host, and logged data is retrieved only after the execution is stopped. Nevertheless, being able to log data allows gathering good enough information about the behavior of the target application. Signal logging becomes a very important feature in polling mode.

Multirate Simulink Models Cannot Be Executed in Multitasking Mode on the Target PC

Because of the polling mode execution scheme, executing Simulink-based target applications in multitasking mode is not possible. The modeling of function-call subsystems to handle asynchronous events (interrupts) is not possible either. This can be a hard restriction, especially for multirate systems. Multirate systems can be executed in single-tasking mode, but because of its sequential execution scheme for all subsystems with different rates, the CPU will most likely overload for the given base sample time. As an important consequence, polling mode is only a feasible alternative to interrupt mode if the model has a single rate or if it can be converted to a single-rate model. A single-rate model implies continuous states only, discrete states only, or mixed continuous and discrete states, if the continuous and discrete subsystems have the same rate. Use the **Format -> Sample time color** feature of Simulink to check for the single rate requirement. Additionally, set the tasking mode property in the **Simulation** menu **Configuration Parameters -> Solver** pane to `SingleTasking` to avoid a possible switch to multitasking mode. For more information on single-tasking mode compared to multitasking mode, see the Real-Time Workshop user's documentation.

I/O Drivers Using Kernel Timing Information Cannot Be Used Within a Model

Some xPC Target drivers use timing information exported from the kernel in order to run properly, for example, for the detection of time-outs. Because the standard timing engine of the real-time kernel is not running during the entire target application execution in polling mode, timing information passed back to the drivers is incorrect. Therefore, you cannot use drivers importing the header file `time_xpcimport.h`. This is a current restriction only. In a future version of polling mode, all drivers will make use of the Pentium counter for getting timing information instead.

Controlling the Target Application

As mentioned, there is no way to interact with the running target application in polling mode. This is especially restrictive for the case of stopping the model execution before the application has reached the stop time that was defined before the execution started. Because polling mode tries to be as optimal as possible, any rapid-prototyping feature except signal logging is disabled. But because I/O driver blocks added to the model are fully functional, you can use I/O drivers to get to a minimal level of interactivity.

Stopping a target application using polling mode — You can use a low-latency digital input driver for the digital PCI board in your model, which reads in a single digital TTL signal. The signal is TTL low unless the model execution should be stopped, for which the signal changes to TTL high. You can connect the output port of the digital input driver block to the input port of a Stop simulation block, found in the standard Simulink block library. This stops the execution of the target application, depending on the state of the digital input signal. You can either use a hardware switch connected to the board-specific input pin or you can generate the signal by other means. For example, you could use another digital I/O board in the host machine and connect the two boards (one in the host, the other in the target) over a couple of wires. You could then use MathWorks Data Acquisition Toolbox to drive the corresponding TTL output pin of the host board to stop the target application execution from within MATLAB.

Generally, you can use the same software/hardware setup for passing other information back and forth during run time of the target application. It is important to understand that any additional feature beside signal logging has to be implemented at the model level, and it is therefore the user's responsibility to optimize for the minimal additional latency the feature introduces. For example, being able to interactively stop the target application execution is paid for by the introduction of an additional 1 μs latency necessary to read the digital signal over the digital I/O board. However, if you need to read digital inputs from the plant hardware anyway, and not all lines are used, you get the feature for free.

Polling Mode Performance

This is preliminary information. All benchmarks have been executed using a 1 GHz AMD Athlon machine, which is the same machine that is at the top of the list displayed by `xpcbench`.

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

The minimum achievable base sample time for model `f14` (type `help xpcbench` for further information in the MATLAB window) using an `ode4` fixed-step integration algorithm is 4 μs with signal logging disabled and 5 μs with signal logging enabled.

A more realistic model, which has been benchmarked, is a second-order continuous controller accessing real hardware over two16 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.

11

xPC Target and Fortran

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

Introduction

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

This section has the following topics:

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

Simulink Demos Directory

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

1 In the MATLAB Command Window, type

```
demos
```

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

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

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

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

The associated Simulink demos page opens.

4 Click **Open this model**.

A library of S-function examples is displayed.

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

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

Prerequisites

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

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

Steps to Incorporate Fortran in Simulink for xPC Target

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

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

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

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

   This step creates the Simulink DLL.

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

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

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

7 Initiate the xPC Target specific Real-Time Workshop build procedure for the demo model. Real-Time Workshop builds and downloads the xPC Target onto the target PC.

Step-by-Step Example of Fortran and xPC Target

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

This section includes the following topics:

Before you start, you should create an xPC Target Simulink model for the Atmosphere model. See "Creating an xPC Target Atmosphere Model for Fortran" on page 11-5.

Creating an xPC Target Atmosphere Model for Fortran

To create an xPC Target Atmosphere model for Fortran, you need to add an xPC Target Scope block to the `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 directory to the working directory, for example, `xpc_fortran_test`.

2 Type

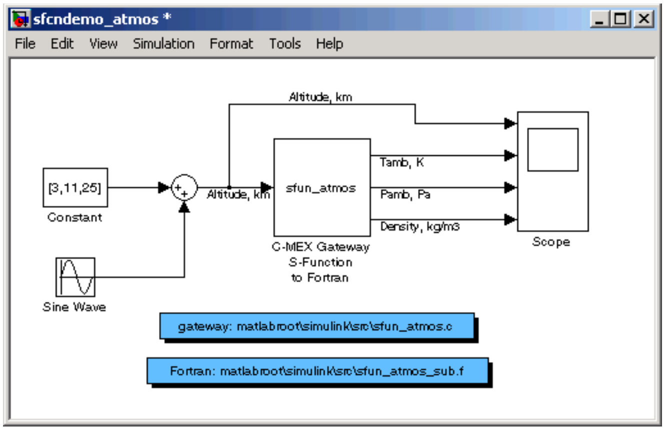
```
sfcndemo_atmos
```

The `sfcndemo_atmos` model is displayed.

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

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

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



5 Double-click the target Scope block.

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

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

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

9 Double-click the Sine Wave block.

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

11 Click **OK**.

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

Your next task is to compile Fortran code. See “Compiling Fortran Files” on page 11-7.

Compiling Fortran Files

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

- DOS command window
- Microsoft Developer Studio IDE

Before you start,

1. Change directory to `<MATLAB root>\simulink\src`.

2. Copy the file `sfun_atmos_sub.f` into your Fortran working directory, for example, `xpc_fortran_test`.

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

Your next task is to compile the Fortran code for xPC Target. See "DOS Command Line" on page 11-7 or "Microsoft Developer Studio IDE" on page 11-8.

DOS Command Line

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

1. Ensure that the system environment has the correct path and variable settings so that you can start the Fortran compiler from the DOS command prompt.

2. From the DOS prompt, change directory to the working directory, for example, `xpc_fortran_test`.

3 Type

```
fl32 -c /iface:cref -G5 -Ox -Zi sfun_atmos_sub.f
```

This command generates the `sfun_atmos_sub.obj` file.

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

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

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

Your next task is to create a wrapper S-function. See “Creating a C-MEX Wrapper S-Function” on page 11-9.

Microsoft Developer Studio IDE

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

1 Define a Fortran project in the Microsoft Developer Studio IDE. This procedure lets the IDE handle the compilation process.

2 Specify at least the `/iface:cref` and `-c` options for the developer studio.

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

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

Your next task is to create a C-MEX wrapper S-function. See “Creating a C-MEX Wrapper S-Function” on page 11-9.

Creating a C-MEX Wrapper S-Function

This section assumes that you have compiled your Fortran code. See "Compiling Fortran Files" on page 11-7.

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

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

1 Change directory to `<MATLAB root>\simulink\src`.

2 Copy the file `sfun_atmos.c` into your Fortran working directory, for example, `xpc_fortran_test`.

Your next task is to compile and link the wrapper S-function. See "Compiling and Linking the Wrapper S-Function" on page 11-9.

Compiling and Linking the Wrapper S-Function

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

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

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

This section assumes that you have created a C_MEX wrapper S-function. See "Creating a C-MEX Wrapper S-Function" on page 11-9.

Invoking the mex command includes the following steps:

1 Compile the wrapper C file sfun_atmos.c. Be sure to link in the following:
   - Compiled wrapper file: sfun_atmos.obj
   - Compiled Fortran code: sfun_atmos_sub.obj
   - Necessary Fortran run-time libraries to resolve external function references and the Fortran run-time environment

2 Type

```
mex -v LINKFLAGS#"$LINKFLAGS dformd.lib dfconsol.lib
dfport.lib" sfun_atmos.c sfun_atmos_sub.obj
```

   Ensure that this whole command is all on one line. This command compiles and links the sfun_atmos_sub.c file. It creates the sfun_atmos.dll file in the same directory.

Note If this command generates a conflict error with libc, you might need to add the option /NODEFAULTLIB:libc.lib to the command. For example, mex -v /NODEFAULTLIB:libc.lib LINKFLAGS#"$LINKFLAGS dformd.lib dfconsol.lib dfport.lib" sfun_atmos.c sfun_atmos_sub.obj.

Your next task is to validate the Fortran code and wrapper S-function. See "Validating the Fortran Code and Wrapper S-Function" on page 11-10.

Validating the Fortran Code and Wrapper S-Function

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

This section assumes that you have compiled and linked a wrapper S-function. See "Compiling and Linking the Wrapper S-Function" on page 11-9.

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

1 At the MATLAB window, type

```
fortran_atmos_xpc
```

This opens the Simulink model associated with the Atmosphere model. This model includes the correct S-function block that is bound to `sfun_atmos.dll`.

2 Select the **Simulation** menu **Start** option to simulate the model.

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

Your next task is to prepare the model to build an xPC Target application. See "Preparing the Model for the xPC Target Application Build" on page 11-11.

Preparing the Model for the xPC Target Application Build

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

- The build procedure has access to `sfun_atmos.sub.obj` for the link stage.
- The build procedure has access to the Fortran run-time libraries (see "Compiling and Linking the Wrapper S-Function" on page 11-9) for the link stage.

This section assumes that you have validated the Fortran code and wrapper S-function (see "Validating the Fortran Code and Wrapper S-Function" on page 11-10).

1 At the MATLAB window, type

```
fortran_atmos_xpc
```

This opens the Simulink model associated with the Atmosphere model.

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

The Configuration Parameters dialog box appears.

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

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

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

5 Click `xpctarget.tlc`.

6 In the **Make command** field, replace `make_rtw` with the following string:

```
make_rtw S_FUNCTIONS_LIB="..\sfun_atmos_sub.obj ..\dfor.lib
..\dfconsol.lib ..\dfport.lib"
```

   Ensure that this whole command is all on one line.

7 Press **Apply**.

8 Press **OK**.

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

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

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

Your next task is to build and run the xPC Target application. See "Building and Running the xPC Target Application" on page 11-13.

Building and Running the xPC Target Application

This section assumes that you have prepared the model to build an xPC Target application. See “Preparing the Model for the xPC Target Application Build” on page 11-11.

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

After the build procedure succeeds, xPC Target automatically downloads the application to the target PC. The Atmosphere model already contains an xPC Target Scope block. This allows you to verify the behavior of the model. You will be able to compare the signals displayed on the target screen with the signals obtained earlier by the Simulink simulation run (see “Validating the Fortran Code and Wrapper S-Function” on page 11-10).

12

Troubleshooting

This chapter describes guidelines, hints, and tips for issues you might encounter while using xPC Target. Refer to The MathWorks Support xPC Target Web site (`http://www.mathworks.com/support/product/XP`) for specific troubleshooting solutions. The xPC Target documentation is also available from this site. This chapter includes the following sections:

General Troubleshooting Hints and Tips

This section lists general troubleshooting tips that you can use as a first attempt to resolve your issues. This section has the following topics:

Is Your Host PC MATLAB Halted?

If your host PC MATLAB halts while creating an xPC Target boot disk,

- Use another formatted disk to create the xPC Target boot disk.
- If your host PC has antivirus software, it might conflict with MATLAB. Disable the software while using MATLAB.
- Verify that the host PC 3.5 inch disk drive is accessible. If it is not accessible, replace the 3.5 inch disk drive.

Is Your Target PC Unable to Boot?

If your target PC cannot boot with the xPC target boot disk,

- Use another formatted disk and create a new xPC Target boot disk.
- Verify that the current properties on the xPC Target boot disk correspond to the environment variables of xPC Target Explorer.

- Verify that the xPC Target boot disk contains files like the following:
  - `BOOTSECT.RTT`
  - `checksum.dat`
  - `XPCTGB1.RTA`

  Note that the name of the last file varies depending on the communication method.
- If any of these files are not present, reinstall xPC Target. This should fix any corrupted files from the previous (initial) installation.
- If problems persist, see "Troubleshooting the Boot Process" in Chapter 3 of the xPC Target getting started documentation.
- If you still cannot boot the target PC, you might need to replace the target PC 3.5 inch disk drive.

Is the Target PC Halted?

If your target PC displays a `System Halted` message while booting,

- Verify that the **TcpIp target driver** parameter is configured correctly in xPC Target Explorer, recreate the xPC Target boot disk, and use that new disk to boot the target PC.
- Ensure that xPC Target supports your target PC hardware. Be sure to verify the network communication hardware.

Is There Communication Between Your PCs?

Use the following MATLAB commands from the host PC to validate the host/target setup:

- `xpctargetping`
- `xpctest`

The `xpctargetping` command performs a basic communication check between the host and target PC. This command only returns `success` if the xPC Target kernel is loaded and is running and the communication between host and target PC is working properly. Use this command for a quick check of the host PC/target PC communications.

The `xpctest` command performs a series of tests on your xPC Target system. These tests range from performing a basic communication check to building and running target applications. At the end of each test, the command returns an `OK` or failure message. If the test is inappropriate for your setup, the command returns a `SKIPPED` message. Use this command for a thorough check of your xPC Target installation.

Communication errors might also occur in the following instances:

- The target PC is running an old xPC Target boot disk that is not in sync with the xPC Target release installed on the host PC. Create a new boot disk for each new release of xPC Target.
- If the communication between the host PC and target PC is TCP/IP, set the host PC 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, The MathWorks is aware of build errors that 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 MATLAB to the firewall exception list.

xPC Target and the Target PC BIOS

The settings of your target PC BIOS will affect your xPC Target. As a general rule, ensure that the host and target PC BIOS have at least the following settings:

- RS-232 communication — If you are using RS-232 communications, ensure that COM ports are enabled for both host and target PCs. Also, ensure through the BIOS that COM1 has a base address of `3F8` and an IRQ of `4`. COM2 must have a base address of `2F8` and an IRQ of `3`. These are the default base address values. Do not change these values.
- Plug-and-Play (PnP) operating system — Disable this feature to ensure that the PCI BIOS sets up the plugged-in PCI cards properly. The xPC Target kernel is not a PnP operating system; you must ensure that this feature is disabled or PCI devices will not work on xPC Target.
- Power Saving modes — Disable all power saving modes.

- USB support — Disable all USB support, including general USB and USB keyboard support. Failure to do this will cause occasional long task execution times (TET).
- PCI boards — Do not detect PCI boards with class code `0xff` in the target PC BIOS. Set this option to `Off` to enable the BIOS to detect and configure all boards.

What PCI Boards Are Installed on Your System?

Use the `getxpcpci` MATLAB command to determine what PCI boards are installed in your xPC Target system. For example,

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

How to Get Updated xPC Target Releases

1. Start **Simulink -> xPC Target -> Product News (Web)**.

2. Look for the section on downloading software and select the version you want.

Are You Working with a New xPC Target Release?

If you are working with a new xPC Target release, either one you download from The MathWorks Web site (`http://www.mathworks.com/web_downloads/`) or one you install from a CD, you must do the following:

- At the MATLAB Command Window, invoke `xpcexplr`.
- Recreate your xPC Target environment (see "Serial Communication" or "Network Communication" in Chapter 2 of the xPC Target getting started documentation).
- Create a new boot disk. Use a new 3.5 inch disk.
- Rebuild target applications on that new xPC Target release.

What Does the Target PC Display?

From the host PC, you can view the target PC monitor with the MATLAB `xpctargetspy` command.

Refer to The MathWorks Support Web Site

This chapter contains general xPC Target troubleshooting tips. Refer to the MathWorks Support xPC Target Web site (`http://www.mathworks.com/support/product/XP`) for more specific troubleshooting solutions. The xPC Target documentation is also available from this site.

Refer to the Documentation

The xPC Target documentation has hints and tips embedded throughout. You should install the Help and PDF documentation to provide easy reference.

- The xPC Target Help documentation is available for installation when you install the xPC Target product either from the CD or Web download.
- The PDF documentation is available for installation from `http://www.mathworks.com`.

Installation, Configuration, and Test Troubleshooting

The following are some issues you might encounter with xPC Target when trying to set up.

Troubleshooting xpctest Results

This section assumes that you have read the "Testing and Troubleshooting the Installation" section of the xPC Target getting started documentation.

xpctest: Test 1 Fails. First, perform the procedure described in the "Test 1, Ping Target System Standard Ping" section of the xPC Target getting started documentation.

Note You can ignore this section if you are using a serial connection. Test 1 is skipped for serial connections.

If you are using a TCP/IP connection and need more help with Test 1, check the following:

- Be sure to use a supported Ethernet card on the target PC. The following Ethernet controllers are supported:
  - Intel 82559 based cards
  - Intel 82559ER based cards
  - Intel 82550 based cards
  - NE2000 based cards
  - AMD lance 79C971(RTLANCE) based cards
  - SMC91C9X based cards

  See "Ethernet Chips Supported by xPC Target" in Chapter 2 of the Getting Started with xPC Target documentation for further details.
- Verify that your hardware is operating correctly. For example, check for faulty network cables and other hardware.
- If you run `xpctest` from a UNC network directory, such as `\\Server\user\work`, a workaround is to change the current MATLAB directory to a local directory and run the test again.

xpctest: Test 2 Fails. First, follow the procedure described in the "Test 2, Ping Target System xPC Target Ping" section of the xPC Target getting started documentation.

If you need more help with Test 2, check the following:

- Use the PC MATLAB command `xpcexplr` to check the environment variables, in particular **Target PC IP address**. If Test 1 passes but Test 2 fails, you might have entered an incorrect IP address.
- If you have a TCP/IP connection, make sure you are using a supported Ethernet card (see "xpctest: Test 1 Fails" on page 12-7).
- For RS-232 connection,
  - Use a null modem cable (see the "Hardware for Serial Communication" section of the Getting Started with xPC Target documentation). If you do not use a null modem cable for an RS-232 connection, communication between the host and target PCs will fail. A null modem cable is shipped with xPC Target.
  - If you do have a null modem cable, check the COM ports on the host and target PC. For example, ensure that the ports are enabled, you have connected the appropriate COM port, and the COM port matches that specified for `xpcexplr`.
  - Ensure that the COM ports on the host and target PCs are enabled in the BIOS. If they are disabled, Test 2 fails.

xpctest: Test 3 Fails. First, follow the procedure described in the "Test 3, Reboot Target Using Direct Call" section of the xPC Target getting started documentation.

If you need more help with Test 3, check the following:

- Did you get the following error?
  - `ReadFile Error: 6`

  Older xPC Target releases might receive this error. This message might occur if the host PC initiates communication with the target PC while the target PC is rebooting, but the kernel on the target PC has not yet loaded. As a workaround, run `xpctest` with the `noreboot` option. For example,

```
xpctest noreboot
```

This command runs the test without trying to reboot the target PC. It displays the following message:

```
### Test 3, Reboot target using direct call: ... SKIPPED
```

- If you directly or indirectly modify the `xpcosc` demo mode that is supplied with xPC Target, Test 3 is likely to fail. To pass this test, restore the original `xpcosc` demo model, using one of the following methods:
  - (Preferred) Download a new copy of the model from the MathWorks FTP site (`ftp://ftp.mathworks.com/pub/tech-support/xpcosc_model/`). Overwrite the old `xpcosc` model with this new one in the directory

    ```
    matlabroot\toolbox\rtw\targets\xpc\xpcdemos
    ```

  - Recreate the original model.
  - Reinstall xPC Target.

Note Do not modify any of the files that are installed with xPC Target. If you want to modify one of these files, copy the file and modify the copy.

---

xpctest: Test 4 Fails. First, follow the procedure described in the “Test 4, Build and Download Application” section of the xPC Target getting started documentation.

If you need more help with Test 4, check the following:

- Verify that a supported compiler is being used.
- If the communication between the host PC and target PC is TCP/IP, set the host PC network interface card (NIC) card and hub to half-duplex mode. Do not set the mode to full-duplex mode.
- Verify the specified path to the supported compiler. You need only the root path to the compiler, not the full path. If you incorrectly specify a path, you might get the following error:

  ```
  Error executing build command: Error using ==> make_rtw
  Error using ==> rtw_c (SetupForVisual)
  Invalid DEVSTUDIO path specified
  ```

  or the following error:

```
Error executing build command: Error using ==> make_rtw
Error using ==> rtw_c
Errors encountered while building model "xpcosc"
```

with the following MATLAB Command Window error:

```
NMAKE: fatal error U1064: MAKEFILE not found and no target
specified
Stop.
```

To correct these errors,

1. Ensure that your compiler is properly installed. For example, all Microsoft Visual compiler components must be in the Microsoft Visual Studio folder after installation.

2. At the MATLAB prompt, type `xpcexplr`. For example,

   ```
   xpcexplr
   ```

3. In the **Select C compiler** field, select the appropriate compiler type (`VisualC` or `Watcom`).

4. In the **Compiler Path** field, enter the root path to the compiler. For example,

   ```
   d:\applications\microsoft visual studio
   ```

   Do not add a terminating back slash (`\`) at the end of the path.

If you still have problems, and you see the following MATLAB Command Window error:

```
ReadFile failed while reading from COM-port
```

1. Check the state of your target PC. If it is unresponsive, you might need to reboot the target PC.

2. In the xPC Target Explorer, try to connect to the target PC again. Be sure to also check the serial connection between the host PC and target PC.

xpctest: Test 5 Fails. This error occurs only when the environment variable settings are out of date.

To correct this, perform the following:

1. At the MATLAB prompt, start **xPC Target Explorer**. For example,

   ```
   xpcexplr
   ```

2. Inspect the environment variables for the problem target PC.
3. If you have xPC Target Embedded Option installed, ensure that, in the **Target boot mode** section, you have selected the `BootFloppy` option.
4. Make necessary changes.
5. Recreate the boot disk. Perform the following:
   a. Insert a formatted floppy disk.
   b. Click **Create Bootdisk**.
6. Reboot the target PC with the boot disk.
7. Rerun `xpctest`.

If this procedure does not resolve the issue, perform the following:

1. At the MATLAB command line, type `updatexpcenv`. For example,

   ```
   updatexpcenv
   ```

2. Recreate the boot disk. Perform the following:
   a. Insert a formatted floppy disk.
   b. At the MATLAB window, type

      ```
      xpcbootdisk
      ```

3. Reboot the target PC.
4. Rerun `xpctest`.

xpctest: Test 6 Fails. This test runs the basic target object constructor, `xpc`. This error rarely occurs without an earlier test failing.

To correct this, perform the following,

1 At the MATLAB command line, refer to and read the `xpc` reference page. For example,

```
help xpc
```

2 Follow any guidance that might be helpful.

3 Reboot the target PC.

4 Rerun and check the results of earlier tests and make the necessary corrections.

xpctest: Test 7 Fails. This test executes a target application (`xpcosc`) on the target PC. This test will fail if you change the `xpcosc` model start time to something other than `0`, such as `0.001`. This change causes the test, and MATLAB itself, to halt. To correct this, set the `xpcosc` model start time back to `0`.

xpctest: Test 8 Fails. This test executes a target application (`xpcosc`) on the target PC. This test might fail if you change the `xpcosc` model (for example, if you remove the Outport block).

To correct this, perform one of the following:

- Set the model back to the original configuration.
- Download a new copy of the model from the MathWorks Web site, depending on the desired version
  - `http://www.mathworks.com/access/pub/xpcosc_model/R12p1version.zip`
  - `http://www.mathworks.com/access/pub/xpcosc_model/R13version.zip`
  - `http://www.mathworks.com/access/pub/xpcosc_model/R14version.zip`

  Overwrite the old `xpcosc` model in the directory

  ```
  matlabroot\toolbox\rtw\targets\xpc\xpcdemos
  ```

- Reinstall xPC Target.

Other issues might also cause this test to fail. If you still need more help, check the following:

- There is a known issue with an earlier version of xPC Target (1.3). It might occur when you run `xpctest` two immediately consecutive times. See the known issue and solution documented in `http://www.mathworks.com/support/solutions/data/27889.html`.
- If you are running a new xPC Target release, be sure that you have a new boot disk for this release. See “Are You Working with a New xPC Target Release?” on page 12-5.
- If you are installing another version of xPC Target on top of an existing version, check the version number of the current installation. At the MATLAB command line, type `xpclib`. The version number appears at the bottom of the Target Simulink block library pop-up window. If the version number is not the one to which you want to upgrade, reinstall xPC Target.

Advanced Troubleshooting

The following are some advanced issues or questions you might have with xPC Target. This section has the following topics:

General I/O Troubleshooting Guidelines

If you encounter issues using the xPC Target I/O drivers,

- Ensure that you have properly configured the driver.
- Ensure that you are using the latest version of xPC Target.
- Test the hardware using the available diagnostic software included with the I/O board from the manufacturer.
- Try a different target PC to verify the behavior.
- Report the issue to The MathWorks Support at (`http://www.mathworks.com/support/contact_us/index.html`).

xPC Target and BIOS

The settings of your host or target PC BIOS will affect your xPC Target results. See "xPC Target and the Target PC BIOS" on page 12-4 for recommended settings. Incorrect BIOS settings can cause questions like the following:

- Why can `getxpcpci` detect PCI boards, but `autosearch -l` cannot?
- Why can my stand-alone application run on some target PCs, but not others?
- Why is my target PC crashing while downloading applications?
- Why is my target PC104 hanging on boot?
- Why is my boot time slow?
- Why is my software not running in real time?

Attempts to Run Any Model Cause CPU Overload Messages on Target PC

This error might occur if you have

- A model sample time that is too small (see "Dealing with Small Model Sample Times" on page 12-15)
- Enabled Advanced Power Management, USB ports in the target PC BIOS, or Plug and Play (PnP) (see "Target PC BIOS" on page 12-16)

Dealing with Small Model Sample Times. If the model has too small a sample time, a CPU overload can occur. This error indicates that to run the target application, the model sample time property requires more CPU time than the sample time for the model (`Fixed step size` property) allows.

When this error occurs, the target object property `CPUoverload` changes from `none` to `detected`. To correct the issue, perform one of the following:

- Change the model `Fixed step size` property to a larger value and rebuild the model. Use the **Solver** node in the Simulink model Configuration Parameters dialog.
- Run the target application on a target PC with a faster processor.

Target PC BIOS. This section assumes that the target PC has BIOS running on it. A CPU overload error can occur if any of the following is enabled:

- Advanced Power Management
- USB ports
- Plug and Play (PnP)

Enabling any of these properties causes non-real-time behavior from the target PC. You must disable these BIOS properties for the target PC to run the target application properly in real time.

More Help. If the preceding procedures do not solve your issue,

- Run `xpcbench` at the MATLAB command line. For example,

  ```
  xpcbench('this')
  ```

  This program accurately evaluates your system. The results indicate the smallest base sample time that an xPC Target application can achieve on your system. For more information on `xpcbench`, type `help xpcbench` at the MATLAB prompt or `http://www.mathworks.com/support/product/XP/productnews/benchmarks.html`.
- Set up the xPC Target environment with a different target PC. Compare the result with the original target PC.

Building a Model That Contains a CAN Board

In releases prior to R14SP1 (xPC Target Version 2.6.1), if you want to use the target PC in a CAN network, you must remember to set up the xPC Target environment for a CAN library. If you do not configure a CAN library into the system, you will get CAN errors when building the target application. In later releases, xPC Target selects the appropriate CAN library for you.

More Help. If the preceding procedures do not resolve the issue, if you can build a target application with the CAN board in your model, but cannot download that application to the target, ensure that

- You are using a supported CAN board.
- You selected the correct choice from the **Can Library** parameter in `xpcexplr`.

Obtaining PCI Board Slot and Bus Information

This section describes how to obtain information about the PCI devices in your xPC Target system. This information is useful if you have or want to use multiple boards of a particular type in your system. Before you start, ensure that the I/O drive supports multiple boards. Refer to one of the following:

- xPC Target I/O reference documentation
- xPC Target Interactive Hardware Selection Guide (`http://www.mathworks.com/support/product/XP/productnews/interactive_guide/xPC_Target_Interactive_Guide.html`)

If the board type supports multiple boards, and these boards are installed in the xPC Target system, perform the following procedure to obtain the bus and slot information for these boards:

1. For example, type

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

2. Note the contents of the `Bus` and `Slot` columns of the PCI devices in which you are interested.

3. Enter the bus and slot numbers as vectors into the **PCI Slot** parameter of the PCI device. For example,

   ```
   [1 9]
   ```

   where `1` is the bus number and `9` is the slot number.

Note You can also obtain PCI device information from **xPC Target Explorer**. To do so, in **xPC Target Explorer**, select a connected target PC, right-click in the **Target Display** pane, and select **View -> Target PC devices**. To return to the default view of target PC properties, select **View -> Target PC Properties**.

For additional information about PCI bus I/O devices, refer to the “PCI Bus I/O Devices” section of the xPC Target I/O reference documentation.

Why is an Error Received While Downloading to the Target PC, But the Host PC Indicates a Successful Download?

If you boot up a target PC with a boot disk from a previous release, then build and download a target application from a host PC running a later release of the xPC Target software, the host PC might indicate a successful download. However, the target PC might display an error message like the following:

```
rt_init timing engine not found
```

This is because the xPC Target software on the boot disk did not match the version of xPC Target software running on the host PC. As a general rule, you must always create a new boot disk with a new xPC Target release or upgrade.

To resolve this, create a new boot disk using the host PC xPC Target software and reboot the target PC with the new boot disk.

Note You should properly label and store old boot disks in case you need to use them again.

Why Does xPC Target Lose Connection with the Host PC When Downloading Some Models?

If you are using xPC Target hardware in a model, downloading the model might cause a loss of communication between the target PC and host PC if either of the following is true:

- The referenced xPC Target board has an initialization time that is too slow.

- The referenced xPC Target driver has an issue.

xPC Target I/O Boards with Slow Initialization Times. Some xPC Target boards have an initialization time that is too slow. This might cause xPC Target to run out of time before a model downloads, causing the host PC to disconnect from the target PC.

By default, if the host PC does not get a response from the target PC after downloading a target application and waiting 5 seconds, the host PC software times out. The target PC responds only after downloading and initializing the target application.

Usually 5 seconds is enough time to initialize a target application, but in some cases it might not be sufficient. The time to download a target application mostly depends on your I/O hardware. For example, thermocouple hardware (such as the PCI-DAS-TC board) takes longer to initialize. With slower hardware, you might also get errors when building and downloading an associated model. Even though the target PC is fine, a false time-out is reported and you might get an error like the following:

```
"cannot connect to ping socket"
```

This is not a fatal error. You can reestablish communication with the following procedure:

1 At the MATLAB Command Window, issue the `xpctargetping` command. For example,

   ```
   xpctargetping
   ```

2 Wait for the system to return from the `xpctargetping`.

3 Restart the target object.

Alternatively, you can enact a more permanent resolution and increase the time-out value using the following procedure:

1 In your MATLAB working directory, create a file called `xpcdltimeout.dat`.

2 In this file, put a single integer. For example, enter

   ```
   20
   ```

In this case, the host PC waits for about 20 seconds before declaring that a time-out has occurred. Note that it does not take 20 seconds for every download. The host PC polls the target PC about once every second, and if a response is returned, declares success. Only in the case where a download really fails does it take the full 20 seconds.

If the file `xpcdltimeout.dat` exists, it is read once before every download. To change the time-out value, change the number in this file. Setting the time-out to `5` is the same as the default. Note also that simply removing the file does not change the time-out to the default value. xPC Target uses the last value you entered until you restart MATLAB.

xPC Target Driver Software Issues. If there really is an error in a driver that causes xPC Target to crash, a time-out occurs and `xpctargetping` fails with an error message. In such an event, you need to reboot the target and reestablish communication between the host PC and target PC.

To get yourself back up and running,

1. Remove the reference to the problem driver from the model.
2. Reboot the target PC.
3. At the MATLAB command line, issue `xpctargetping` to reestablish communications.
4. If the driver with which you are having problems is one provided by The MathWorks, try to pinpoint the problem area (for example, determine whether certain settings in the driver block cause problems).

Alternatively, you can exit and restart MATLAB.

Why Is My Requested xPC Target Sample Time Different from the Measured Sample Time?

You might notice that the sample time you request does not equal the actual sample time you measure from your model. This difference depends on your hardware. For xPC Target, your model sample time is as close to your requested time as the hardware allows.

However, hardware does not allow infinite precision in setting the spacing between the timer interrupts. It is this limitation that can cause the divergent sample times.

For all PCs, the only timer that can generate interrupts is based on a 1.193 MHz clock. For xPC Target, the timer is set to a fixed number of ticks of this frequency between interrupts. If you request a sample time of 1/10000, or 100, microseconds, you do not get exactly 100 ticks. Instead, xPC Target calculates that number as

$$100 \text{ x } 10^{-6} \text{ seconds X } 1.193 \text{ x } 10^{6} \text{ ticks/seconds} = 119.3 \text{ ticks}$$

xPC Target rounds this number to the nearest whole number, 119 ticks. The actual sample time is then

$$119 \text{ ticks}/(1.193 \text{ X } 10^{6} \text{ ticks/second}) = 99.75 \text{ X } 10^{-6} \text{ seconds}$$
$$(99.75 \text{ microseconds})$$

Compared to the requested original sample time of 100 microseconds, this value is .25% faster.

As an example of how you can use this value to derive the expected deviation for your hardware, assume the following:

- Output board that generates a 50 Hz sine wave (expected signal)
- Sample time of 1/10000
- Measured signal of 50.145 Hz

The difference between the expected and measured signals is .145, which deviates from the expected signal value by .29% (.145/50). Compared to the previously calculated value of .25%, there is a difference of .04% from the expected value.

If you want to further refine the measured deviation for your hardware, assume the following:

- Output board that generates a 50 Hz sine wave (expected signal)
- Sample time of 1/10200
- Measured signal of 50.002 Hz

$$1/10200 \text{ seconds X } 1.193 \text{ x } 10^{6} \text{ ticks/seconds} = 116.96 \text{ ticks}$$

Round this number to the nearest whole number of 117 ticks. The resulting frequency is then

```
(116.96 ticks/117)(50) = 49.983 Hz
```

The difference between the expected and measured signal is .019, which deviates from the expected signal value by .038% (.019/50.002). The deviation when the sample time is 1/10000 is .04%.

Some amount of error is common for most PCs, and the margin of error varies from machine to machine.

Note Most high-level operating systems, like Windows or Linux, occasionally insert extra long intervals to compensate for errors in the timer. Be aware that xPC Target does not attempt to compensate for timer errors. This is because for xPC Target, close repeatability is more important for most models than exact timing. However, some chips might have inherent designs that produce residual jitters that could affect your system. For example, some Pentium chips might produce residual jitters on the order of .5 microsecond from interrupt to interrupt.

Why Did I Get Error -10: Invalid File ID on the Target PC?

You might get this error if you are acquiring signal data with a scope of type `file`. This error occurs because the size of the signal data file exceeds the available space on the disk. The signal data will most likely be corrupted and irretrievable. You should delete the signal data file and reboot the xPC Target system. To prevent this occurrence, monitor the size of the signal data file as the scope acquires data.

Refer to The MathWorks Support xPC Target Web site (`http://www.mathworks.com/support/product/XP`) for additional information.

Can I Write Custom xPC Target Device Drivers?

You might want to write your own driver if you want to include an unsupported device driver in your xPC Target system. The xPC Target documentation does not currently describe how to write your own xPC Target device drivers. Refer to the following for further information:

- `http://www.mathworks.com/support/solutions/data/31528.html`
- Existing xPC Target device driver source code. Refer to the following directory:

  ```
  MATLABROOT\toolbox\rtw\targets\xpc\target\build\xpcblocks
  ```

- In the `include` directory of the device driver source code area, pay particular attention to the following files:
  - `io_xpcimport.h`
  - `pci_xpcimport.h`
  - `time_xpcimport.h`

Before you consider writing custom device drivers for the xPC Target system, you should possess

- Good C programming skills
- Knowledge of writing S-functions and compiling those functions as C-MEX functions
- Knowledge of SimStruct, a MATLAB Simulink C language header file that defines the Simulink data structure and the SimStruct access macros. It encapsulates all the data relating to the model or S-function, including block parameters and outputs.
- An excellent understanding of the I/O hardware. Because of the real-time nature of xPC Target, you must develop drivers with minimal latency. And since most drivers access the I/O hardware at the lowest possible level (register programming), you must have a good understanding of how to control the board with register information. Indirectly, this means that you must have access to the register-level programming manual for the device.
- A good knowledge of port and memory I/O access over various buses. You need this information to access I/O hardware at the register level.

Can I Create a Stand-Alone xPC Target Application to Interact with a Target Application?

Yes. You can use either the xPC Target API dynamic link library (DLL) or the xPC Target component object model (COM) API library to create custom stand-alone applications to control a real-time application running on the target PC. To deploy these stand-alone applications, you must have the xPC Target Embedded Option. Without the xPC Target Embedded Option, you can create and use the stand-alone application in your environment, but cannot deploy that application on another host PC that does not contain your licensed copy of xPC Target.

See the xPC Target API documentation for details.

Can Signal Outputs from Virtual Blocks Be Tagged?

You cannot directly tag signal outputs from virtual blocks. Instead, do the following:

1 Add a unity gain block (a Gain block with a gain of 1) to the model.

2 Connect the signal output of the virtual block to the input of the unity gain block.

3 Tag the output signal of the unity gain block.

Why Has the Stop Time Changed?

If you change 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, Real Time Workshop calculates a number of steps based on the current step size and stop time. If the stop time is not an integral multiple of the step size, Real Time Workshop also adjusts the stop time for that model based on the original stop time and step size. If you later change a step size for a target application, but do not recompile the code, xPC Target uses the new step size and the adjusted stop time. This might lead to fewer steps than you expect.

For example, if a model has a stop time of 2.4 and a step size of 1, Real-Time Workshop adjusts the stop time of the model to 2 at compilation. If you change the step size to .6 but do not recompile the code, the expected number of steps is 4, but the actual number of steps is 3. This is because Real-Time Workshop still uses the adjusted stop time of 2.

To avoid this problem, ensure that the original stop time (as specified in the model) is an integral multiple of the original step size.

Why Do I Get File System Disabled Error?

If your target PC does not have a FAT hard disk, the monitor on the target PC displays the following error:

```
ERROR -4: drive not found
No accessible disk found: file system disabled
```

If you do not want to access the target PC file system, you can ignore this message. If you want to access the target PC file system, add a FAT hard disk to the target PC system and reboot.

Why Does the getparamid Function Return Nothing?

The `getparamid` and `getsignalid` functions accept `block_name` parameters. For these functions, enter for `block_name` the mangled name that Real-Time Workshop 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.

How Can I Diagnose Network Problems with xPC Target?

If you experience network problems when using xPC Target, refer to The MathWorks Support xPC Target Web site (`http://www.mathworks.com/support/product/XP`). This Web site has the most up-to-date information about this topic.

Target PC Command-Line Interface Reference

xPC Target provides a limited set of commands that you can use to work the target application after it has been loaded to the target PC, and to interface with the scopes for that application. This chapter is a reference of those commands.

Target PC Commands

The target PC command-line interface enables you to work with target and scope objects in a limited capacity. Methods let you interact directly with the scope or target. Property commands let you work with target and scope properties. Variable commands let you alias target PC command-line interface commands to names of your choice. This section provides references for the following:

- "Target Object Methods" on page 13-3
- "Target Object Property Commands" on page 13-3
- "Scope Object Methods" on page 13-5
- "Scope Object Property Commands" on page 13-8
- "Aliasing with Variable Commands" on page 13-10

Refer to Chapter 6, "Using the Target PC Command-Line Interface," for a description of how to use these methods and commands.

Target Object Methods

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

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

Target PC Command	Description and Syntax	MATLAB Equivalent
`start`	Start the target application currently loaded on the target PC. Syntax: `start`	`tg.start` or `+tg`
`stop`	Stop the target application currently running on the target PC. Syntax: `stop`	`tg.stop` or `-tg`
`reboot`	Reboot the target PC. Syntax: `reboot`	`tg.reboot`

Target Object Property Commands

When you are using the target PC command-line interface, target object properties are limited to parameters, signals, stop time, and sample time. Note the difference between a parameter index (0, 1, . . .) and a parameter name (P0, P1, . . .).

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

Target PC Command	Description and Syntax	MATLAB Equivalent
`getpar`	Display the value of a block parameter using the parameter index. Syntax: `getpar parameter_index`	`get(tg, 'parameter_name')`
`setpar`	Change the value of a block parameter using the parameter index. Syntax: `setpar parameter_index = floating_point_number`	`set(tg, 'parameter_name', number)`
`stoptime`	Enter a new stop time. Use `inf` to run the target application until you manually stop it or reset the target PC. Syntax: `stoptime = floating_point_number`	`tg.stoptime = number`

`sampletime`	Enter a new sample time.
	
	Syntax: `sampletime = floating_point_number`
		`tg.sampletime = number`
		
		`set(tg, 'SampleTime', number)`

Target PC Command	Description and Syntax	MATLAB Equivalent
`P#`	Display or change the value of a block parameter. For example, `P2` or `P2=1.23e-4`. Syntax: `parameter_name` or `parameter_name = floating_point_number` `parameter_name` is `P0, P1, . . .`	`tg.getparam(parameter_index)` `tg.setparam(parameter_index,floating_point_number)`
`S#`	Display the value of a signal. For example, `S2`. Syntax: `signal_name` `signal_name` is `S0, S1, . . .`	`tg.getsignal(signal_index)`

Scope Object Methods

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

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

Target PC Command	Description and Syntax	MATLAB Equivalent
addscope	addscope scope_index addscope	tg.addscope(scope_index) tg.addscope
remscope	remscope scope_index remscope all	tg.remscope(scope_index) tg.remscope
startscope	startscope scope_index	sc.start or +sc
stopscope	stopscope scope_index	sc.stop or -sc
addsignal	addsignal scope_index = signal_index1, signal_index2, . . .	sc.addsignal(signal_ index_vector)
remsignal	remsignal scope_index = signal_index1, signal_index2, . . .	sc.remsignal(signal_ index_vector)
viewmode	Zoom in to one scope or zoom out to all scopes. Syntax: viewmode scope_index or left-click the scope window viewmode 'all' or right-click any scope window Press the function key for the scope, and then press **V** to toggle viewmode.	

Target PC Command	Description and Syntax	MATLAB Equivalent
ylimit	ylimit scope_index ylimit scope_index = auto ylimit scope_index = num1, num2	
grid	grid scope_index on grid scope_index off	

Scope Object Property Commands

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

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

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

Target PC	MATLAB
`numsamples scope_index = number`	`sc.NumSamples = number`
`decimation scope_index = number`	`sc.Decimation = number`
`scopemode scope_index = 0 or numerical, 1 or redraw, 2 or sliding, 3 or rolling`	`sc.Mode = 'numerical', 'redraw', 'sliding', 'rolling'`
`triggermode scope_index = 0, freerun, 1, software, 2, signal, 3, scope`	`sc.TriggerMode = 'freerun', 'software', 'signal', 'scope'`
`numprepostsamples scope_index = number`	`sc.NumPrePostSamples = number`
`triggersignal scope_index = signal_index`	`sc.TriggerSignal = signal_index`
`triggersample scope_index = number`	`sc.TriggerSample = number`
`triggerlevel scope_index = number`	`sc.TriggerLevel = number`

Target PC	MATLAB
`triggerslope scope_index = 0, either, 1, rising, 2, falling`	`sc.TriggerSlope = 'Either', 'Rising', 'Falling'`
`triggerscope scope_index2 = scope_index1`	`sc.TriggerScope = scope_index1`
`triggerscopesample scope_index= integer`	`sc.TriggerSample = integer`
Press the function key for the scope, and then press **S** or move mouse into the scope window.	`sc.trigger`

Aliasing with Variable Commands

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

Target PC Command	Description and Syntax	MATLAB Equivalent
`setvar`	Set a variable to a value. Later you can use that variable to do a macro expansion. Syntax: `setvar variable_name = target_pc_command` For example, you can type `setvar aa=startscope 2`, `setvar bb=stopscope 2`.	None
`getvar`	Display the value of a variable. Syntax: `getvar variable_name`	None
`delvar`	Delete a variable. Syntax: `delvar variable_name`	None
`delallvar`	Delete all variables. Syntax: `delallvar`	None
`showvar`	Display a list of variables. Syntax: `showvar`	None

14

Function Reference

Functions — Categorical List

This topic contains reference pages for xPC Target functions.

Software Environment

`getxpcenv`	List environment properties assigned to MATLAB variable
`setxpcenv`	Change xPC Target environment properties
`updatexpcenv`	Change current environment properties to new properties
`xpcbootdisk`	Create xPC Target boot disk and confirm current environment properties
`xpcwwwenable`	Disconnect target PC from current client application

GUI

`xpcexplr`	Open xPC Target Explorer window
`xpctargetspy`	Open **Real-Time xPC Target Spy** window on host PC

Test

`getxpcpci`	Determine which PCI boards are installed in target PC
`xpctargetping`	Test communication between host and target computers
`xpctest`	Test the xPC Target installation

Target Objects

The target object methods allow you to control a target application on the target PC from the host PC. You enter target object methods in the MATLAB window on the host PC or use M-file scripts. To access the M-file help for these methods, use the syntax

```
help xpctarget.xpc/method_name
```

If you want to control the target application from the target PC, use target PC commands. See "Using the Target PC Command-Line Interface" on page 6-1.

`xpc`	Call target object constructor, `xpctarget.xpc`
`xpctarget.xpc`	Create a target object representing target application
`delete`	Remove target object
`set (target object)`	Change target object property values
`get (target object)`	Return target object property values

`start (target object)`	Start execution of target application on target PC
`stop (target object)`	Stop execution of target application on target PC
`load`	Download target application to target PC
`unload`	Remove current target application from target PC
`addscope`	Create one or more scopes
`setparam`	Change writable target object parameters
`getparam`	Return value of target object parameter index
`getparamid`	Get parameter index from parameter list
`getparamname`	Get block path and parameter name from index list
`getscope`	Get scope object pointing to scope defined in kernel
`getsignal`	Return value of target object signal index
`getsignalid`	Get signal index or signal property from signal list
`getsignalname`	Get signal name from index list
`remscope`	Remove a scope from the target PC
`saveparamset`	Save current target application parameter values
`loadparamset`	Restore parameter values saved in specified file
`getlog`	Get all or part of output logs from target object
`reboot`	Reboot target PC

`close`	Close serial port connecting host PC with target PC.
`targetping`	Test communication between a host and its target computers

Scope Objects

The scope object methods allow you to control scopes on your target PC.

If you want to control the target application from the target PC, use target PC commands. See Chapter 6, “Using the Target PC Command-Line Interface.”

`set (scope object)`	Change property values for scope objects
`get (scope object)`	Return property values for scope objects
`addsignal`	Add signals to a scope represented by a scope object
`remsignal`	Remove signals from a scope represented by a scope object
`start (scope object)`	Start execution of a scope on target PC
`stop (scope object)`	Stop execution of a scope on target PC
`trigger`	Software-trigger start of data acquisition for one or more scopes

File and File System Objects

xPC Target uses the `xpctarget.ftp` object to represent a target PC file, such as a signal data file created by an xPC Target scope of type `file`. Use this object to perform file transfer operations on that file. xPC Target uses the `xpctarget.fs` object to represent the target PC file system. Use this object to perform file system manipulations.

Both `xpctarget.ftp` and `xpctarget.fs` belong to the `xpctarget.fsbase` object. This object encompasses the methods common to `xpctarget.ftp` and `xpctarget.fs`. xPC Target creates the `xpctarget.fs` base object when you create either an `xpctarget.ftp` or `xpctarget.fs` object.

This section includes the following topics:

- “xpctarget.fsbase Functions” on page 14-6 — List of methods with a brief description
- “xpctarget.ftp Functions” on page 14-7 — List of methods with a brief description
- “xpctarget.fs Functions” on page 14-7 — List of methods with a brief description

It concludes with a more complete description of these methods.

Refer to Chapter 7, “Working with Target PC Files and File Systems,” for a description of how to use these objects and methods.

xpctarget.fsbase Functions

You enter `xpctarget.fsbase` object methods in the MATLAB Command Window on the host PC or use M-file scripts.

You can call the `xpctarget.fsbase` methods for both `xpctarget.ftp` and `xpctarget.fs` objects.

`xpctarget.fs`	Create xPC Target file system object
`cd`	Change directory on target PC
`dir`	List contents of current directory on target PC
`mkdir`	Make a directory on target PC

`pwd`	Retrieve current directory path of target PC
`rmdir`	Remove directory from target PC

xpctarget.ftp Functions

The `xpctarget.ftp` methods allow you to work with a target PC file, such as a signal data file, from the host PC. You enter target object methods in the MATLAB window on the host PC or use M-file scripts.

`xpctarget.ftp`	Create xPC Target FTP object
`get (ftp)`	Retrieve copy of requested file from target PC
`put`	Copy file from host PC to target PC

xpctarget.fs Functions

The `xpctarget.fs` methods allow you to work with the target PC file system from the host PC. You enter target object methods in the MATLAB window on the host PC or use M-file scripts.

The `xpctarget.fs` methods are listed in the following table.

`diskinfo`	Get information about target PC drive
`fclose`	Close one or more open target PC files
`fileinfo`	Get target PC file information
`filetable`	Get information about open files in target PC file system
`fopen`	Open target PC file for reading
`fread`	Read open target PC file
`fwrite`	Write binary data to the open target PC file
`getfilesize`	Get size of file on target PC
`removefile`	Remove file from target PC

Functions — Alphabetical List

This section contains function reference pages listed alphabetically.

Purpose

Create one or more scopes

Syntax

MATLAB command line

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

```
addscope(target_object, scope_type, scope_number)
target_object.addscope(scope_type, scope_number)
```

Create a scope, scope object, and assign to a MATLAB variable

```
scope_object = addscope(target_object, scope_type, scope_number)

scope_object = target_object.addscope(scope_type, scope_number)
```

Target PC command line — When you are using this command on the target PC, you can only add a scope of type `target`.

```
addscope
addscope scope_number
```

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. xPC Target supports 10 scopes of scopes of type target and host, and eight scopes of type file, 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.

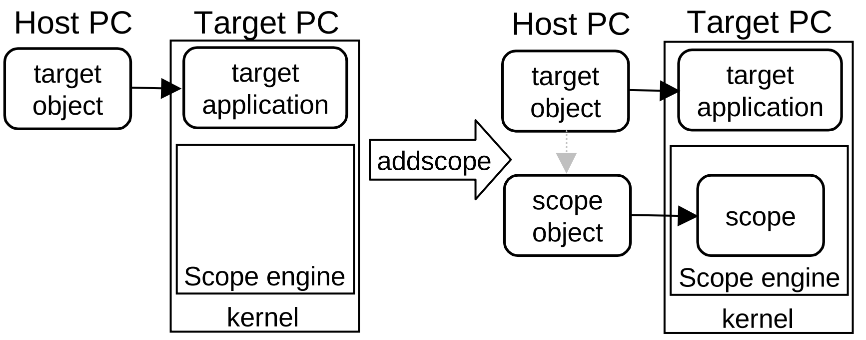
addscope

A scope acquires data from the target application and displays that data on the target PC, uploads the data to the host PC, or stores that data in a file in the target PC file system.

All scopes of type `target`, `host`, or `file` run on the target PC.

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

Scope of type host — Collects data and waits for a command from the host PC for uploading the data. The data is then displayed using a scope viewer on the host or other MATLAB functions.



Scope of type file — Data collected is stored in a file in the target PC file system. You can then transfer the data to another PC for examination or plotting.

Examples

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

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

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

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

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

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

Create a scope and scope object `sc4 of` type `file` using the method `addscope`. A file scope is created on the target PC with an index of 4. A scope object is created on the host PC and is assigned to the variable `sc4`. The target object property `Scopes` is changed from `No scopes defined` to `4`.

`sc4 = addscope(tg,'file',4)` or `sc4 = tg.addscope('file',4)`

See Also

xPC Target target object methods `remscope` and `getscope`.

xPC Target M-file demo scripts listed in "xPC Target Demos" on page 5-8.

Purpose

Add signals to a scope represented by a scope object

Syntax

MATLAB command line

```
addsignal(scope_object_vector, signal_index_vector)
```

```
scope_object_vector.addsignal(signal_index_vector)
```

Target command line

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

Arguments

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

Description

`addsignal` adds signals to a scope object. The signals must be specified by their indices, which you can retrieve using the target object method `getsignalid`. If the `scope_object_vector` has two or more scope objects, the same signals are assigned to each scope.

Note You must stop the scope before you can add a signal to it.

Examples

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

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

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

```
sc1.Signals
```

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

Another way to add signals without using the method `addsignal` is to use the scope object method `set`.

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

Or, to directly assign signal values to the scope object property `Signals`,

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

See Also

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

Purpose Change directory on target PC

Syntax **MATLAB command line**

```
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 PC directory to change to.

Description Method of xpctarget.fsbase, xpctarget.ftp, and xpctarget.fs objects. From the host PC, changes directory on the target PC.

Examples Change directory from the current to one named logs for the file system object fsys.

cd(fsys,logs) or fsys.cd(logs)

Change directory from the current to one named logs for the FTP object f.

cd(f,logs) or f.cd(logs)

See Also xPC Target file object methods mkdir and pwd.

MATLAB cd function.

Purpose Close serial port connecting host PC with target PC

Syntax **MATLAB command line**

```
close(target_object)
target_object.close
```

Arguments

`target_object`	Name of a target object.

Description `close` closes the serial connection between the host PC and a target PC. If you want to use the serial port for another function without quitting MATLAB — for example, a modem — use this function to close the connection.

delete

Purpose

Remove target object

Syntax

MATLAB command line

```
delete(target_object)
target_object.delete
```

Arguments

target_object	Name of a target object.

Description

Use this method to completely remove the target object. If there are any scopes still associated with the target, this method removes all those scope objects as well.

To ensure that you have successfully removed a target object, type

```
target_object
```

If a message like the following is displayed, you have successfully removed the target object.

```
target_object =

    handle
```

See Also

The xPC Target target object methods `targetping` and `xpctarget.xpc`.

Purpose List contents of current directory on target PC

Syntax **MATLAB command line**

```
dir(file_obj)
```

Arguments

`file_obj`	Name of the `xpctarget.ftp` or `xpctarget.fs` object.

Description Method of `xpctarget.fsbase`, `xpctarget.ftp`, and `xpctarget.fs` objects. From the host PC, lists the contents of the current directory on the target PC.

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 directory, shown as a cell array. The name, stored in the first element of the cell array, can have up to eight characters. The three-character file extension is stored in the second element of the cell array.
- `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 directory

Examples List the contents of the current directory for the file system object `fsys`. You can also list the contents of the current directory 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 directory.

See Also

xPC Target file object methods `mkdir`, `cd`, and `pwd`.

MATLAB `dir` function.

Purpose

Get information about target PC drive

Syntax

MATLAB command line

```
diskinfo(filesys_obj,target_PC_drive)
filesys_obj.diskinfo(target_PC_drive)
```

Arguments

`filesys_obj`	Name of the `xpctarget.fs` file system object.
`target_PC_drive`	Name of the target PC drive for which to return information.

Description

Method of `xpctarget.fs` objects. From the host PC, returns disk information for the specified target PC drive.

Examples

Return disk information for the target PC 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
```

Purpose

Close one or more open target PC files

Syntax

MATLAB command line

```
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 PC, closes one or more open files in the target PC file system (except standard input, output, and error). The `file_ID` argument is the file identifier associated with an open file (see `fopen` and `filetable`). You cannot have more than eight files open in the file system.

Examples

Close the open file identified by the file identifier `h` in the file system object `fsys`.

`fclose(fsys,h)` or `fsys.fclose(h)`

See Also

xPC Target file object methods `fopen`, `fread`, `filetable`, and `fwrite`.

MATLAB `fclose` function.

Purpose Calculate parameter values for Fastcom 422/2-PCI board

Syntax **MATLAB command line**

```
[a b ] = fc422mexcalcbits(frequency)
[a b df] = fc422mexcalcbits(frequency)
```

Arguments

`frequency`	Desired baud rate for the board

`[a b] = fc422mexcalcbits(frequency)` accepts a baud rate (in units of baud/second) and converts this value into two parameters `a b`. You must enter these values for the parameter **Clock bits** of the Fastcom 422/2-PCI driver clock. The desired baud rate (`frequency`) must range between 30e3 and 1.5e6, which is a hardware limitation of the clock circuit.

`[a b df] = fc422mexcalcbits(frequency)` accepts a baud rate (in units of baud/second) and converts this value into two parameters `a b`. You must enter these values for the parameter **Clock bits** of the Fastcom 422/2-PCI driver block. The third value, `df`, indicates the actual baud rate that is created by the generated parameters `a b`. The clock circuit has limited resolution and is unable to perfectly match an arbitrary frequency. The desired baud rate (`frequency`) must range between 30e3 and 1.5e6, which is a hardware limitation of the clock circuit.

fileinfo

Purpose Get target PC file information

Syntax **MATLAB command line**

```
fileinfo(filesys_obj,file_ID)
filesys_obj.fileinfo(file_ID)
```

Arguments

`filesys_obj`	Name of the `xpctarget.fs` file system object.
`file_ID`	File identifier of the file for which to get file information.

Description Method of `xpctarget.fs` objects. From the host PC, gets the information for the file associated with `file_ID`.

Examples Return file information for the file associated with the file identifier `h` in the file system object `fsys`.

`fileinfo(fsys,h)` or `fsys.fileinfo(h)`

```
ans =
              FilePos: 0
        AllocatedSize: 12288
        ClusterChains: 1
   VolumeSerialNumber: 1.0450e+009
             FullName: 'C:\DATA.DAT'
```

Purpose

Get information about open files in target PC file system

Syntax

MATLAB command line

```
filetable(filesys_obj)
filesys_obj.filetable
```

Arguments

`filesys_obj`	Name of the `xpctarget.fs` file system object.

Description

Method of `xpctarget.fs` objects. From the host PC, displays a table of the open files in the target PC file system. You cannot have more than eight files open in the file system.

Examples

Return a table of the open files in the target PC file system for the file system object `fsys`.

`filetable(fsys)` or `fsys.filetable`

```
ans =
Index    Handle  Flags     FilePos  Name
------------------------------------------
    0  00060000  R__          8512  C:\DATA.DAT
    1  00080001  R__             0  C:\DATA1.DAT
    2  000A0002  R__          8512  C:\DATA2.DAT
    3  000C0003  R__          8512  C:\DATA3.DAT
    4  001E000S  R__             0  C:\DATA4.DAT
```

The table returns the open file handles in hexadecimal. To convert a handle to one that other `xpctarget.fs` methods, such as `fclose`, can use, use the `hex2dec` function. For example,

```
h1 = hex2dec('001E0001'))

h1 =
1966081
```

To close that file, use the `xpctarget.fs` `fclose` method. For example,

```
fsys.fclose(h1);
```

See Also

xPC Target file object methods `fopen` and `fclose`.

Purpose

Open target PC file for reading

Syntax

MATLAB command line

```
file_ID = fopen(file_obj,'file_name')
file_ID = file_obj.fopen('file_name')
file_ID = fopen(file_obj,'file_name',permission)
file_ID = file_obj.fopen('file_name',permission)
```

Arguments

`file_obj`	Name of the `xpctarget.fs` object.
`'file_name'`	Name of the target PC to open.
`permission`	Values are `'r'` or `'w'`. This argument is optional with `'r'` as the default value.

Description

Method of `xpctarget.fs` objects. From the host PC, opens the specified filename on the target PC for binary access.

The permission argument values are

- `'r'` to open the file for reading (default)
- `'w'` to open the file for writing. 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 `fclose`, `fread`, and `fwrite`).

Examples

Open the file `data.dat` in the target PC file system object `fsys`. Assign the resulting file handle to a variable for reading.

```
h = fopen(fsys,'data.dat') or fsys.fopen('data.dat')

ans =
     2883584
d = fread(h);
```

See Also

xPC Target file object methods `fclose`, `fread`, and `fwrite`.

MATLAB `fopen` function.

Purpose

Read open target PC file

Syntax

MATLAB command line

```
fread(file_obj,file_ID)
file_obj.fread(file_ID)
```

Arguments

`file_obj`	Name of the `xpctarget.fs` object.
`file_ID`	File identifier of the file to read.

Description

Method of `xpctarget.fs` objects. From the host PC, reads the binary data from the file on the target PC and writes it into matrix A. The `file_ID` argument is the file identifier associated with an open file (see `fopen`).

Examples

Open the file `data.dat` in the target PC file system object `fsys`. Assign the resulting file handle to a variable for reading.

```
h = fopen(fsys,'data.dat') or fsys.fopen('data.dat')

ans =
     2883584
d = fread(h);
```

This reads the file `data.dat` and stores the contents of the file to `d`. This content is in the xPC Target file format.

See Also

xPC Target file object methods `fclose`, `fopen`, and `fwrite`.

MATLAB `fread` function.

fwrite

Purpose

Write binary data to the open target PC file

Syntax

MATLAB command line

```
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 PC, writes the elements of matrix `A` to the file identified by `file_ID`. The data is written to the file in column order. The `file_ID` argument is the file identifier associated with an open file (see `fopen`). `fwrite` requires that the file be open with write permission.

Examples

Open the file `data.dat` in the target PC 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

xPC Target file object methods `fclose`, `fopen`, and `fread`.

MATLAB `fwrite` function.

Purpose

Retrieve copy of requested file from target PC

Syntax

MATLAB command line

```
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 PC.

Description

Method of `xpctarget.ftp` objects. Copies the specified filename from the target PC to the current directory of the host PC. `file_name` must be either a fully qualified filename on the target PC, or located in the current directory of the target PC.

Examples

Retrieve a copy of the file named `data.dat` from the current directory of the target PC file object `f`.

`get(f,'data.dat')` or `f.get('data.dat')`

```
ans = data.dat
```

See Also

xPC Target file object methods `put`.

get (scope object)

Purpose Return property values for scope objects

Syntax **MATLAB command line**

```
get(scope_object_vector)
```

```
get(scope_object_vector, 'scope_object_property')
```

```
get(scope_object_vector, scope_object_property_vector)
```

Arguments

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

Description `get` gets the value of readable scope object properties from a scope object or the same property from each scope object in a vector of scope objects. Scope object properties let you select signals to acquire, set triggering modes, and access signal information from the target application. You can view and change these properties using scope object methods.

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

Property	Description	Writable
`Application`	Name of the Simulink model associated with this scope object.	
`AutoRestart`	For scopes of type `'File'`, enable the file scope to collect data up to the number of samples (`NumSamples`), then start over again, appending the new data to the end of the signal data file. Clear the **AutoRestart** check box to have the scope of type `'File'` collect data up to **Number of samples**, then stop. If the named signal data file already exists when you start the target application, xPC Target overwrites the old data with the new signal data. For scopes of type `'Host'` or `'Target'`, this parameter has no effect.	Yes
`Data`	Contains the output data for a single data package from a scope. For scopes of type `'Target'` or `'File'`, this parameter has no effect.	
`Decimation`	A number `n`, where every `n`th sample is acquired in a scope window.	Yes

Property	Description	Writable
`Filename`	Provide a name for the file to contain the signal data. By default, the target PC writes the signal data to a file named `C:\data.dat` for scope blocks. Note that for scopes of type `'File'` created through the MATLAB interface, there is no name initially assigned to `FileName`. After you start the scope, xPC Target assigns a name for the file to acquire the signal data. This name typically consists of the scope object name, `ScopeId`, and the beginning letters of the first signal added to the scope. For scopes of type `'Host'` or `'Target'`, this parameter has no effect.	Yes
`Grid`	Values are `'on'` and `'off'`. For scopes of type `'Host'` or `'File'`, this parameter has no effect.	Yes

Property	Description	Writable
`Mode`	For scopes of type `'Target'`, indicate how a scope displays the signals. Values are `'Numerical'`, `'Redraw'` (default), `'Sliding'`, and `'Rolling'`. For scopes of type `File`, specify when a file allocation table (FAT) entry is updated. Values are `'Lazy'` or `'Commit'`. Both modes write the signal data to the file. With `'Commit'` mode, each file write operation simultaneously updates the FAT entry for the file. This mode is slower, but the file system always knows the actual file size. With `'Lazy'` mode, the FAT entry is updated only when the file is closed and not during each file write operation. This mode is faster, but if the system crashes before the file is closed, the file system might not know the actual file size (the file contents, however, will be intact). For scopes of type `Host`, this parameter has no effect.	Yes
`NumPrePostSamples`	For scopes of type `'Host'` or `'Target'`, this parameter is the number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set `TriggerMode` to `'FreeRun'`, this property has no effect on data acquisition.	

Property	Description	Writable
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 scopes of type 'File', 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.	
Signals	List of signal indices from the target object to display on the scope.	
StartTime	Time within the total execution time when a scope begins acquiring a data package. For scopes of type 'Target', this parameter has no effect.	

Property	Description	Writable
`Status`	Indicate whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are `'Acquiring'`, `'Ready for being Triggered'`, `'Interrupted'`, and `'Finished'`.	
`Time`	Contains the time data for a single data package from a scope.	
`TriggerLevel`	If `TriggerMode` is `'Signal'`, indicates the value the signal has to cross to trigger the scope and start acquiring data. The trigger level can be crossed with either a rising or falling signal.	Yes
`TriggerMode`	Trigger mode for a scope. Valid values are `'FreeRun'` (default), `'Software'`, `'Signal'`, and `'Scope'`.	Yes

Property	Description	Writable
TriggerSample	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.	
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

Property	Description	Writable
`TriggerSlope`	If `TriggerMode` is `'Signal'`, indicates whether the trigger is on a rising or falling signal. Values are `'Either'` (default), `'Rising'`, and `'Falling'`.	Yes
`Type`	Determines whether the scope is displayed on the host computer or on the target computer. Values are `'Host'`, `'Target'`, and `'File'`.	
`WriteSize`	Enter the block size, in bytes, of the data chunks. This parameter specifies that a memory buffer, of length number of samples (`NumSamples`), collect data in multiples of `WriteSize`. By default, this parameter is 512 bytes, which is the typical disk sector size. Using a block size that is the same as the disk sector size provides optimal performance. If you experience a system crash, you can expect to lose an amount of data the size of `WriteSize`. For scopes of type `'Host'` or `'Target'`, this parameter has no effect.	Yes
`YLimit`	Minimum and maximum *y*-axis values. This property can be set to `'auto'`. For scopes of type `'Host'` or `'File'`, this parameter has no effect.	Yes

Examples

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

```
get(sc)
```

List the value for the scope object property `Type`. Notice that the property name is a string, in quotation marks, and is not case sensitive.

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

See Also

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

Purpose

Return target object property values

Syntax

MATLAB command line

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

Arguments

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

Description

`get` gets the value of readable target object properties from a target object.

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

Property	Description	Writable
`Application`	Name of the Simulink model and target application built from that model.	
`AvgTET`	Average task execution time. This value is an average of the measured CPU times, in seconds, to run the model equations and post outputs during each sample interval. Task execution time is nearly constant, with minor deviations due to cache, memory access, interrupt latency, and multirate model execution.	
`Connected`	Communication status between the host PC and the target PC. Values are `'Yes'` and `'No'`.	
`CPUoverload`	CPU status for overload. If the target application requires more CPU time than the sample time of the model, this value is set from `'none'` to `'detected'` and the current run is stopped. Correcting `CPUoverload` requires either a faster processor or a larger sample time.	

Property	Description	Writable
`ExecTime`	Execution time. Time, in seconds, since your target application started running. When the target application stops, the total execution time is displayed.	
`LogMode`	Controls which data points are logged: • 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 located at **Simulation** menu **Configuration Parameters -> xPC Target options** pane.	
`MaxTET`	Maximum task execution time. Corresponds to the slowest time (longest time measured), in seconds, to update model equations and post outputs.	
`MinTET`	Minimum task execution time. Corresponds to the fastest time (smallest time measured), in seconds, to update model equations and post outputs.	

Property	Description	Writable
`Mode`	Type of Real-Time Workshop code generation. Values are `'Real-Time Singletasking'`, `'Real-Time Multitasking'`, and `'Accelerate'`. The default value is `'Real-Time Singletasking'`.	
	Even if you select `'Real-Time Multitasking'`, the actual mode can be `'Real-Time Singletasking'`. This happens if your model contains only one or two tasks and the sample rates are equal.	
`NumLogWraps`	The number of times the circular buffer wrapped. The buffer wraps each time the number of samples exceeds `MaxLogSamples`.	
`NumParameters`	The number of parameters from your Simulink model that you can tune or change.	
`NumSignals`	The number of signals from your Simulink model that are available to be viewed with a scope.	
`OutputLog`	Storage in the MATLAB workspace for the output or y-vector logged during execution of the target application.	

Property	Description	Writable
`Parameters`	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.	
`SampleTime`	Time between samples. This value equals the step size, in seconds, for updating the model equations and posting the outputs.	Yes
`Scopes`	List of index numbers, with one index for each scope.	
`SessionTime`	Time since the kernel started running on your target PC. This is also the elapsed time since you booted the target PC. Values are in seconds.	
`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

Property	**Description**	**Writable**
`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.	
`StateLog`	Storage in the MATLAB workspace for the state or x-vector logged during execution of the target application.	
`Status`	Execution status of your target application. Values are `'stopped'` and `'running'`.	
`StopTime`	Time when the target application stops running. Values are in seconds. The original value is set in the **Simulation** menu **Configuration Parameters** dialog. When the `ExecTime` reaches the `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 located at **Simulation** menu **Configuration Parameters -> xPC Target options** pane.	

Property	Description	Writable
TimeLog	Storage in the MATLAB workspace for the time or t-vector logged during execution of the target application.	
ViewMode	Display either all scopes or a single scope on the target PC. Value is 'all' or a single scope index. This property is active only if the environment property TargetScope is set to enabled.	Yes

Examples

List the value for the target object property StopTime. Notice that the property name is a string, in quotation marks, and not case sensitive.

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

```
ans = 0.2
```

See Also

The xPC Target target object method set (target object).

The scope object methods get (scope object) and set (target object).

The built-in MATLAB functions get and set.

Purpose Get size of file on target PC

Syntax **MATLAB command line**

```
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 PC, gets the size (in bytes) of the file identified by the `file_ID` file identifier on the target PC file system.

Examples Get the size of the file identifier `h` for the file system object `fsys`.

`getfilesize(fsys,h)` or `fsys.getfilesize(h)`

getlog

Purpose Get all or part of output logs from target object

Syntax **MATLAB command line**

```
log = getlog(target_object, 'log_name', first_point,
number_samples, decimation)
```

Arguments

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

Description Use this function instead of the function `get` when you want only part of the data.

Examples To get the first 1000 points in a log,

```
Out_log = getlog(tg, 'TETLog', 1, 1000)
```

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

```
Output_log = getlog(tg, 'TETLog', 1, ,2)
Time_log = getlog(tg, 'TimeLog', 1, ,2)
plot(Time_log, Output_log)
```

See Also xPC Target target object method `get (target object)`.

The procedure “Entering the Real-Time Workshop Parameters” on page 3-42.

Purpose Return value of target object parameter index

Syntax

MATLAB command line

```
getparam(target_object, parameter_index)
```

Arguments

`target_object`	Name of a target object. The default name is `tg`.
`parameter_index`	Index number of the parameter.

Description `getparam` returns the value of the parameter associated with `parameter_index`.

Examples Get the value of parameter index `5`.

```
getparam(tg, 5)
ans = 400
```

getparamid

Purpose Get parameter index from parameter list

Syntax **MATLAB command line**

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

Arguments

`target_object`	Name of a target object. The default name is `tg`.
`'block_name'`	Simulink block path without model name.
`'parameter_name'`	Name of a parameter within a Simulink block.

Description `getparamid` returns the index of a parameter in the parameter list based on the path to the parameter name. The names must be entered in full and are case sensitive. Note, enter for `block_name` the mangled name that Real-Time Workshop 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 The xPC Target scope object method `getsignalid`.

The xPC target M-file demo scripts listed in “xPC Target Demos” on page 5-8.

Troubleshooting chapter question “Why Does the getparamid Function Return Nothing?” on page 12-25.

Purpose Get block path and parameter name from index list

Syntax **MATLAB command line**

```
getparamname(target_object, parameter_index)
```

Arguments

target_object	Name of a target object. The default name is tg.
parameter_index	Index number of the parameter.

Description getparamname returns two argument strings, block path and parameter name, from the index list for the specified parameter index.

Examples Get the block path and parameter name of parameter index 5.

```
[blockPath,parName]=getparamname(tg,5)
blockPath =

Signal Generator

parName =

Amplitude
```

getscope

Purpose Get scope object pointing to scope defined in kernel

Syntax

MATLAB command line

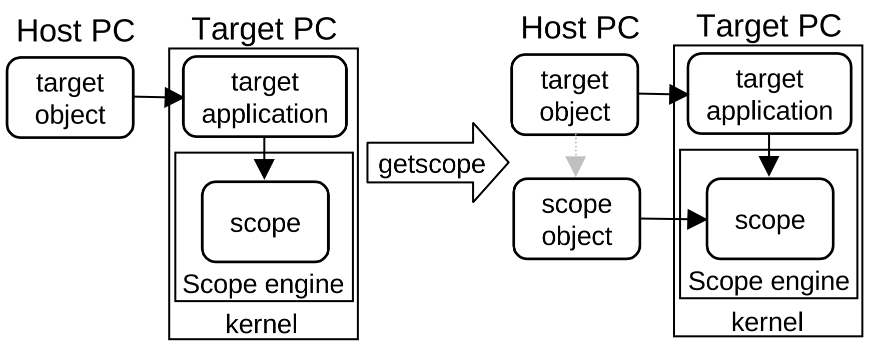
```
scope _object_vector = getscope(target_object, scope_number)

scope_object = target_object.getscope(scope_number)
```

Arguments

`target_object`	Name of a target object.
`scope_number_vector`	Vector of existing scope indices listed in the target object property `Scopes`. The vector can have only one element.
`scope_object`	MATLAB variable for a new scope object vector. The vector can have only one scope object.

Description

`getscope` returns a scope object vector. If you try to get a nonexistent scope, the result is an error. You can retrieve the list of existing scopes using the method `get(target_object, 'scopes')` or `target_object.scopes`.



Examples

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

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

The following example gets the properties of all scopes on the target PC and creates a vector of scope objects on the host PC. If the target object has more than one scope, it create a vector of scope objects.

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

See Also

xPC Target target object methods `getxpcenv` and `remscope`.

xPC target M-file demo scripts listed in “xPC Target Demos” on page 5-8.

getsignal

Purpose Return value of target object signal index

Syntax

MATLAB command line

```
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 Get signal index or signal property from signal list

Syntax **MATLAB command line**

```
getsignalid(target_object, 'signal_name')
tg.getsignalid('signal_name')
```

Arguments

`target_object`	Name of an existing target object.
`signal_name`	Enter the name of a signal from your Simulink model. For blocks with a single signal, the `signal_name` is equal to the `block_name`. For blocks with multiple signals, xPC Target appends S1, S2 ... to the `block_name`.

Description `getsignalid` returns the index or name of a signal from the signal list, based on the path to the signal name. The block names must be entered in full and are case sensitive. Note, enter for `block_name` the mangled name that Real-Time Workshop 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 xPC Target target object method `getparamid`.

xPC Target M-file demo scripts listed in "xPC Target Demos" on page 5-8.

Troubleshooting chapter question "Why Does the getparamid Function Return Nothing?" on page 12-25.

getsignalname

Purpose

Get signal name from index list

Syntax

MATLAB command line

```
getsignalname(target_object, signal_index)
```

Arguments

`target_object`	Name of a target object. The default name is `tg`.
`signal_index`	Index number of the signal.

Description

`getparamname` returns one argument string, signal name, from the index list for the specified signal index.

Examples

Get the signal name of signal ID 2.

```
[sigName]=getsignalname(tg,2)
sigName =

Gain2
```

Purpose

List environment properties assigned to MATLAB variable

Syntax

MATLAB Command Line

```
getxpcenv
```

Description

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

The environment properties define communication between the host PC and target PC, the type of C compiler and its location, and the type of target boot floppy created during the setup process. You can view these properties using the `getxpcenv` function or the xPC Target Explorer. An understanding of the environment properties will help you to correctly configure the xPC Target environment.

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

Environment Property	Description
`TargetRAMSizeMB`	Values are `'Auto'` and `'Manual'`. From the xPC Target Explorer window **Target RAM size** list, select either `Auto` or `Manual`. If you select `Manual`, enter the amount of RAM, in megabytes, installed on the target PC. This property is set by default to `Auto`. **Target RAM size** defines the total amount of installed RAM in the target PC. This RAM is used for the kernel, target application, data logging, and other functions that use the heap. If **Target RAM size** is set to `Auto`, the target application automatically determines the amount of memory up to 64 MB. If the target PC does not contain more than 64 MB of RAM, or you do not want to use more than 64 MB, select `Auto`. If the target PC has more than 64 MB of RAM, and you want to use more than 64 MB, select `Manual`, and enter the amount of RAM installed in the target PC.
`MaxModelSize`	Values are `'1MB'`, `'4MB'`, and `'16MB'`. From the xPC Target Explorer window **Maximum model size** list, select either `1 MB`, `4 MB`, or `16 MB`. Choosing the maximum model size reserves the specified amount of memory on the target PC for the target application. The remaining memory is used by the kernel and by the heap for data logging. 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.

Environment Property	Description
`HostTargetComm`	Values are `'RS232'` and `'TcpIp'`. From the xPC Target Explorer window **Host target communication** list, select either `RS232` or `TCP/IP`. If you select `RS232`, you also need to set the property `RS232HostPort`. If you select `TCP/IP`, then you also need to set all properties that start with `TcpIp`.
`RS232HostPort`	Values are `'COM1'` and `'COM2'`. From the xPC Target Explorer window **Host port** list, select either `COM1` or `COM2` for the connection on the host computer. xPC Target automatically determines the COM port on the target PC. Before you can select an RS-232 port, you need to set the `HostTargetComm` property to `RS232`.
`RS232Baudrate`	Values are `'115200'`, `'57600'`, `'38400'`, `'19200'`, `'9600'`, `'4800'`, `'2400'`, and `'1200'`. From the **Baud rate** list, select `115200`, `57600`, `38400`, `19200`, `9600`, `4800`, `2400`, or `1200`.
`TcpIpTargetAddress`	Value is `'xxx.xxx.xxx.xxx'`. In the xPC Target Explorer window **Target PC IP address** box, enter a valid IP address for your target PC. Ask your system administrator for this value. For example, `192.168.0.1`.

Environment Property	Description
`TcpIpTargetPort`	Value is `'xxxxx'`. In the xPC Target Explorer window **TcpIp target port** box, enter a value greater than `20000`. This property is set by default to `22222` and should not cause any problems. The number is higher than the reserved area (`telnet`, `ftp`, ...) and it is only of use on the target PC.
`TcpIpSubNetMask`	Value is `'xxx.xxx.xxx.xxx'`. In the xPC Target Explorer window **LAN subnet mask address** text box, enter the subnet mask of your LAN. Ask your system administrator for this value. For example, your subnet mask could be `255.255.255.0`.
`TcpIpGateway`	Value is `'xxx.xxx.xxx.xxx'`. In the xPC Target Explorer window **TcpIp gateway address** box, enter the IP address for your gateway. This property is set by default to `255.255.255.255`, which means that a gateway is not used to connect to the target PC. If you communicate with your target PC from within a LAN that uses gateways, and your host and target computers are connected through a gateway, then you need to enter a value for this property. If your LAN does not use gateways, you do not need to change this property. Ask your system administrator.

Environment Property	Description
`TcpIpTargetDriver`	Values are `'NE2000'`, `'SMC91C9X'`, `'I82559'`, and `'RTLANCE'`. From the xPC Target Explorer window **TcpIp target driver** list, select `NE2000`, `SMC91C9X`, `I82559`, or `RTLANCE`. The Ethernet card provided with xPC Target uses the `NE2000` driver.
`TcpIpTargetBusType`	Values are `'PCI'` and `'ISA'`. From the xPC Target Explorer window **TcpIp target bus type** list, select either `PCI` or `ISA`. This property is set by default to `PCI`, and determines the bus type of your target PC. You do not need to define a bus type for your host PC, which can be the same or different from the bus type in your target PC. 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
`TcpIpTargetISAMemPort`	Value is `'0xnnnn'`. If you are using an ISA bus Ethernet card, you must enter values for the properties `TcpIpISAMemPort` and `TcpIpISAIRQ`. The values of these properties must correspond to the jumper settings or ROM settings on your ISA bus Ethernet card. On your ISA bus card, assign an IRQ and I/O port base address by moving the jumpers on the card. Set the I/O port base address to around 0x300. If one of these hardware settings leads to a conflict in your target PC, choose another I/O port base address and make the corresponding changes to your jumper settings.
`TcpIpTargetISAIRQ`	Value is `'n'`, where *n* is between 4 and 15. If you are using an ISA bus Ethernet card, you must enter values for the properties `TcpIpISAMemPort` and `TcpIpISAIRQ`. The values of these properties must correspond to the jumper settings or ROM settings on the ISA-bus Ethernet card. On your ISA bus card, assign an IRQ and I/O-port base address by moving the jumpers on the card. The MathWorks recommends setting the IRQ to 5, 10, or `11`. If one of these hardware settings leads to a conflict in your target PC, choose another IRQ and make the corresponding changes to your jumper settings.

<table>
<tr><th>Environment Property</th><th>Description</th></tr>
<tr><td><code>TargetScope</code></td><td>Values are <code>'Disabled'</code> and <code>'Enabled'</code>.

From the xPC Target Explorer window Enable target scope list, select either <code>Enabled</code> or <code>Disabled</code>.

The property <code>TargetScope</code> is set by default to <code>Enabled</code>. If you set <code>TargetScope</code> to <code>Disabled</code>, the target PC displays information as text.

To use all the features of the target scope, you also need to install a keyboard and mouse on the target PC.</td></tr>
<tr><td><code>TargetMouse</code></td><td>Values are <code>'None'</code>, <code>'PS2'</code>, <code>'RS232 COM1'</code>, and <code>'RS232 COM2'</code>.

From the xPC Target Explorer window Target mouse list, select <code>None</code>, <code>PS2</code>, <code>RS232 COM1</code>, or <code>RS232 COM2</code>.

Before you can select a target mouse, you need to set the <code>Target Scope</code> property to <code>Enabled</code>.

Target mouse allows you to disable or enable mouse support on the target PC:

• If you do not connect a mouse to the target PC, you need to set this property to <code>None</code>; otherwise, the target application might not behave properly.
• If the target PC supports PS/2 devices (keyboard and mouse) and you connect a PS/2 mouse, set this property to <code>PS2</code>.
• If you connect a serial RS-232 mouse to the target PC, select either <code>RS232 COM1</code> or <code>RS232 COM2</code> depending on which serial port you attached the mouse to.</td></tr>
</table>

Environment Property	Description
`EmbeddedOption`	Values are `'Disabled'` and `'Enabled'`. This property is read only. Note that the xPC Target Embedded Option is enabled only if you purchase an additional license.
`TargetBoot`	Values are `'BootFloppy'`, `'DOSLoader'`, and `'StandAlone'`. From the xPC Target Explorer window **Target boot mode** list, select `BootFloppy`, `DOSLoader`, or `StandAlone`. If your license file does not include the license for the xPC Target Embedded Option, the **Target boot mode** box is disabled, with `BootFloppy` as your only selection. With the xPC Target Embedded Option licensed and installed, you have the additional choices of `DOSLoader` and `StandAlone`.

Examples

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

```
env = getxpcenv

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

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

```
env = getxpcenv
```

See Also

xPC Target functions `setxpcenv`, `updatexpcenv`, and `xpcbootdisk`

Purpose Determine which PCI boards are installed in target PC

Syntax

MATLAB Command Line

```
getxpcpci(target_object, 'type_of_boards')
```

Arguments

`target_object`	Variable name to reference the target object.
`type_of_boards`	Values are no arguments, `'all'`, and `'supported'`.

Description

The `getxpcpci` function displays, in the MATLAB window, which PCI boards are installed in the target PC. By default, `getxpcpci` displays this information for the target object, tg. If you have multiple target PCs in your system, you can call the `getxpcpci` function for a particular target object, `target_object`.

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

For a successful query,

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

Examples

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

```
pcidevs = getxpcpci
```

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

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

Display a list of the currently supported PCI devices in the xPC Target block library. The result is displayed.

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

When called with the `'supported'` option, `getxpcpci` does not access the target PC.

To display the list of PCI devices installed on the target PC, `tg1`, first create a target object, `tg1`, for that target PC. Then, call getxpcpci with the `'all'` option. For example

```
tg1=xpctarget.xpc('RS232','COM1','115200')
getxpcpci(tg1, 'all')
```

Purpose

Download target application to target PC

Syntax

MATLAB command line

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

Arguments

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

Description

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

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

Examples

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

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

See Also

xPC Target function `unload`.

xPC Target M-file demo scripts listed in “xPC Target Demos” on page 5-8.

loadparamset

Purpose Restore parameter values saved in specified file

Syntax **MATLAB command line**

```
loadparamset(target_object,'filename')
target_object.loadparamset('filename')
```

Arguments

`target_object`	Name of an existing target object.
`filename`	Enter the name of the file that contains the saved parameters.

Description `loadparamset` restores the target application parameter values saved in the file `filename`. This file must be located on a local drive of the target PC. This method assumes that you have a parameter file from a previous run of the `saveparamset` method.

See Also xPC Target target object method `saveparamset`.

Purpose Make a directory on target PC

Syntax **MATLAB command line**

```
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 directory to be created.

Description Method of `xpctarget.fsbase`, `xpctarget.ftp`, and `xpctarget.fs` objects. From the host PC, makes a new directory in the current directory on the target PC file system.

Note that to delete a directory from the target PC, you need to reboot the PC into DOS or some other operating system and use a utility in that system to delete the directory.

Examples Create a new directory, `logs`, in the target PC file system object `fsys`.

`mkdir(fsys,logs)` or `fsys.mkdir(logs)`

Create a new directory, `logs`, in the target PC FTP object `f`.

`mkdir(f,logs)` or `f.mkdir(logs)`

See Also xPC Target file object methods `dir` and `pwd`.

MATLAB `mkdir` function.

Purpose

Copy file from host PC to target PC

Syntax

MATLAB command line

```
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 PC.

Description

Method of `xpctarget.ftp` objects. Copies a file from the host PC to the target PC. `file_name` must be a file in the current directory of the host PC. The method writes `file_name` to the target PC 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 directory of the host PC to the current directory of the target PC FTP object `f`.

`put(f,'data2.dat')` or `fsys.put('data2.dat')`

See Also

xPC Target file object methods `dir` and `get (ftp)`.

Purpose Retrieve current directory path of target PC

Syntax **MATLAB command line**

```
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 PC directory.

Examples Return the target PC current directory for the file system object `fsys`.

`pwd(fsys)` or `fsys.pwd`

Return the target PC current directory for the FTP object `f`.

`pwd(f)` or `f.pwd`

See Also xPC Target file object methods `dir` and `mkdir`.

MATLAB `pwd` function.

reboot

Purpose Reboot target PC

Syntax

MATLAB command line

```
reboot(target_object)
```

Target PC command line

```
reboot
```

Arguments

`target_object`	Name of an existing target object.

Description

`reboot` reboots the target PC, and if a target boot disk is still present, the xPC target kernel is reloaded.

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

Note This method might not work on some target hardware.

See Also

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

Purpose

Remove scope from target PC

Syntax

MATLAB command line

```
remscope(target_object, scope_number_vector)
target_object.remscope(scope_number_vector)

remscope(target_object)
target_object.remscope
```

Target PC command line

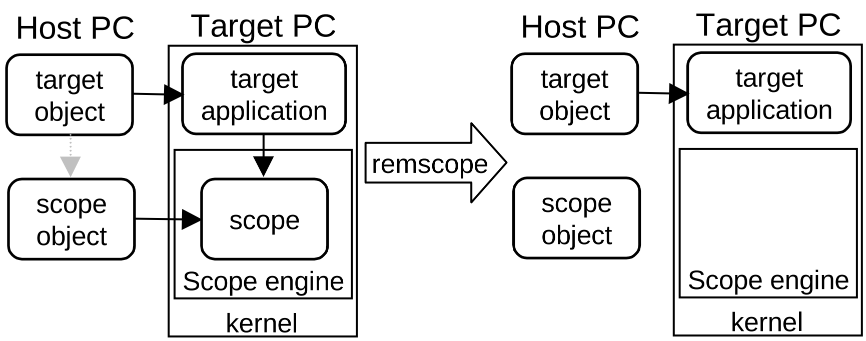
```
remscope scope_number
remscope 'all'
```

Arguments

`target_object`	Name of a target object. The default name is `tg`.
`scope_number_vector`	Vector of existing scope indices listed in the target object property `Scopes`.
`scope_number`	Single scope index.

Description

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



Note that you can only permanently remove scopes that are added with the method `addscope`. This is a scope that is outside a model. If you remove a scope that has been added through a scope block (the scope block is inside the model), a subsequent run of that model creates the scope again.

Examples

Remove a single scope.

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

Remove two scopes.

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

Remove all scopes.

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

See Also

xPC Target target object methods `addscope` and `getscope`.

xPC target M-file demo scripts listed in “xPC Target Demos” on page 5-8.

Purpose Remove file from target PC

Syntax **MATLAB command line**

```
removefile(file_obj,file_name)
file_obj.removefile(file_name)
```

Arguments

`file_name`	Name of the file to remove from the target PC file system.
`file_obj`	Name of the `xpctarget.fs` object.

Description Method of `xpctarget.fs` objects Removes a file from the target PC file system.

Note You cannot recover this file once it is removed.

Examples Remove the file `data2.dat` from the target PC file system `fsys`.

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

remsignal

Purpose Remove signals from a scope represented by a scope object

Syntax

MATLAB command line

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

Target command line

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

Arguments

`scope_object`	MATLAB object created with the target object method `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
```

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

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

See Also

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

rmdir

Purpose

Remove directory from target PC

Syntax

MATLAB command line

```
rmdir(file_obj,dir_name)
file_obj.rmdir(dir_name)
```

Arguments

`dir_name`	Name of the directory to remove from the target PC file system.
`file_obj`	Name of the `xpctarget.fs` object.

Description

Method of `xpctarget.fsbase`, `xpctarget.ftp`, and `xpctarget.fs` objects. Removes a directory from the target PC file system.

Note You cannot recover this directory once it is removed.

Examples

Remove the directory `data2dir.dat` from the target PC file system `fsys`.

`rmdir(f,'data2dir.dat')` or `fsys.rmdir('data2dir.dat')`

Purpose

Save current target application parameter values

Syntax

MATLAB command line

```
saveparamset(target_object,'filename')
target_object.saveparamset('filename')
```

Arguments

`target_object`	Name of an existing target object.
`filename`	Enter the name of the file to contain the saved parameters.

Description

`saveparamset` saves the target application parameter values in the file `filename`. This method saves the file on a local drive of the target PC (C:\ by default). You can later reload these parameters with the `loadparamset` function.

You might want to save target application parameter values if you change these parameter values while the application is running in real time. Saving these values enables you to easily recreate target application parameter values from a number of application runs.

See Also

xPC Target target object method `loadparamset`.

set (scope object)

Purpose Change property values for scope objects

Syntax **MATLAB command line**

```
set(scope_object_vector)

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

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

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

Arguments

`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.

Property	Description	Writable
`Application`	Name of the Simulink model associated with this scope object.	
`AutoRestart`	For scopes of type `'File'`, enable the file scope to collect data up to the number of samples (`NumSamples`), then start over again, appending the new data to the end of the signal data file. Clear the **AutoRestart** check box to have the scope of type `'File'` collect data up to **Number of samples**, then stop. If the named signal data file already exists when you start the target application, xPC Target overwrites the old data with the new signal data. For scopes of type `'Host'` or `'Target'`, this parameter has no effect.	Yes
`Data`	Contains the output data for a single data package from a scope. For scopes of type `'Target'` or `'File'`, this parameter has no effect.	
`Decimation`	A number `n`, where every `n`th sample is acquired in a scope window.	Yes

Property	Description	Writable
`Filename`	Provide a name for the file to contain the signal data. By default, the target PC writes the signal data to a file named `C:\data.dat` for scope blocks. Note that for scopes of type `'File'` created through the MATLAB interface, there is no name initially assigned to `FileName`. After you start the scope, xPC Target assigns a name for the file to acquire the signal data. This name typically consists of the scope object name, `ScopeId`, and the beginning letters of the first signal added to the scope. For scopes of type `'Host'` or `'Target'`, this parameter has no effect.	Yes
`Grid`	Values are `'on'` and `'off'`. For scopes of type `'Host'` or `'File'`, this parameter has no effect.	Yes

Property	Description	Writable
Mode	For scopes of type 'Target', indicate how a scope displays the signals. Values are 'Numerical', 'Redraw' (default), 'Sliding', and 'Rolling'. For scopes of type File, specify when a file allocation table (FAT) entry is updated. Values are 'Lazy' or 'Commit'. Both modes write the signal data to the file. With 'Commit' mode, each file write operation simultaneously updates the FAT entry for the file. This mode is slower, but the file system always knows the actual file size. With 'Lazy' mode, the FAT entry is updated only when the file is closed and not during each file write operation. This mode is faster, but if the system crashes before the file is closed, the file system might not know the actual file size (the file contents, however, will be intact). For scopes of type Host, this parameter has no effect.	Yes
NumPrePostSamples	For scopes of type 'Host' or 'Target', this parameter is the number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set TriggerMode to 'FreeRun', this property has no effect on data acquisition.	

Property	Description	Writable
`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 scopes of type `'File'`, 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.	
`Signals`	List of signal indices from the target object to display on the scope.	
`StartTime`	Time within the total execution time when a scope begins acquiring a data package. For scopes of type `'Target'`, this parameter has no effect.	

Property	Description	Writable
`Status`	Indicate whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are `'Acquiring'`, `'Ready for being Triggered'`, `'Interrupted'`, and `'Finished'`.	
`Time`	Contains the time data for a single data package from a scope.	
`TriggerLevel`	If `TriggerMode` is `'Signal'`, indicates the value the signal has to cross to trigger the scope and start acquiring data. The trigger level can be crossed with either a rising or falling signal.	Yes
`TriggerMode`	Trigger mode for a scope. Valid values are `'FreeRun'` (default), `'Software'`, `'Signal'`, and `'Scope'`.	Yes

Property	Description	Writable
TriggerSample	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.	
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

Property	Description	Writable
TriggerSlope	If TriggerMode is 'Signal', indicates whether the trigger is on a rising or falling signal. Values are 'Either' (default), 'Rising', and 'Falling'.	Yes
Type	Determines whether the scope is displayed on the host computer or on the target computer. Values are 'Host', 'Target', and 'File'.	
WriteSize	Enter the block size, in bytes, of the data chunks. This parameter specifies that a memory buffer, of length number of samples (NumSamples), collect data in multiples of WriteSize. By default, this parameter is 512 bytes, which is the typical disk sector size. Using a block size that is the same as the disk sector size provides optimal performance. If you experience a system crash, you can expect to lose an amount of data the size of WriteSize. For scopes of type 'Host' or 'Target', this parameter has no effect.	Yes
YLimit	Minimum and maximum *y*-axis values. This property can be set to 'auto'. For scopes of type 'Host' or 'File', this parameter has no effect.	Yes

Examples

Get a list of writable properties for a scope object.

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

ans=
```

```
            NumSamples: {}
            Decimation: {}
           TriggerMode: {5x1 cell}
         TriggerSignal: {}
          TriggerLevel: {}
          TriggerSlope: {4x1 cell}
          TriggerScope: {}
         TriggerSample: {}
               Signals: {}
     NumPrePostSamples: {}
                  Mode: {5x1 cell}
                YLimit: {}
                  Grid: {}
```

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

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

See Also

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

Purpose Change target 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 PC command line - Commands are limited to the target object properties `stoptime`, `sampletime`, and parameters.

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

Arguments

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

Description `set` sets the properties of the target object. Not all properties are user writable.

Properties must be entered in pairs or, using the alternate syntax, as one-dimensional cell arrays of the same size. This means they must both be row vectors or both column vectors, and the corresponding values for properties in `property_name_vector` are stored in `property_value_vector`. The writable properties for a target object are listed in the following table. This table includes a description of the properties:

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
`SampleTime`	Time between samples. This value equals the step size, in seconds, for updating the model equations and posting the outputs.	Yes
`ShowParameters`	Flag set to view or hide the list of parameters from your Simulink blocks. This list is shown when you display the properties for a target object. Values are `'on'` and `'off'`.	Yes
`ShowSignals`	Flag set to view or hide the list of signals from your Simulink blocks. This list is shown when you display the properties for a target object. Values are `'on'` and `'off'`.	Yes

Property	Description	Writable
`StopTime`	Time when the target application stops running. Values are in seconds. The original value is set in the **Simulation** menu **Configuration Parameters** dialog. When the `ExecTime` reaches the `StopTime`, the application stops running.	Yes
`ViewMode`	Display either all scopes or a single scope on the target PC. Value is `'all'` or a single scope index. This property is active only if the environment property `TargetScope` is set to `enabled`.	Yes

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

Examples

Get a list of writable properties for a scope object.

```
set(tg)

ans =

         StopTime: {}
       SampleTime: {}
         ViewMode: {}
          LogMode: {}
   ShowParameters: {}
      ShowSignals: {}
```

Change the property `ShowSignals` to on.

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

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

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

See Also

xPC Target target object method `get (target object)`.

Scope object methods `get (scope object)` and `set (scope object)`.

Built in MATLAB functions `get` and `set`.

xPC target M-file demo scripts listed in "xPC Target Demos" on page 5-8.

Purpose Change writable target object parameters

Syntax **MATLAB command line**

```
setparam(target_object, 'parameter_value')
```

Arguments

`target_object`	Name of an existing target object. The default name is `tg`.
`parameter_value`	Value for a target object parameter.

Description Method of a target object. Set the value of the target parameter. This method returns a structure that stores the parameter index, previous parameter values, and new parameter values in the following fields:

- `parIndexVec`
- `OldValues`
- `NewValues`

Examples Set the value of parameter index `5` to `100`.

```
setparam(tg, 5, 100)
ans =

parIndexVec: 5
OldValues: 400
NewValues: 100
```

Purpose Change xPC Target environment properties

Syntax **MATLAB Command Line**

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

Arguments

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

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

The environment properties define communication between the host PC and target PC, the type of C compiler and its location, and the type of target boot floppy created during the setup process. With the exception of the `Version` property, you can set these properties using the `xpcexplr` function or the xPC Target Explorer window. An understanding of the environment properties will help you to correctly configure the xPC Target environment.

Environment Property	Description
`Version`	xPC Target version number. Read only.
`CCompiler`	Values are `'Watcom'` and `'VisualC'`. From the xPC Target Explorer window compiler list, select either `Watcom` or `VisualC`.

Environment Property	Description
`CompilerPath`	Value is a valid compiler root directory. Enter the path where you installed a Watcom C/C++ or Microsoft Visual C/C++ compiler. If the path is invalid or the directory does not contain the compiler, an error message appears when you use the function `updatexpcenv` or build a target application.
`TargetRAMSizeMB`	Values are `'Auto'` and `'Manual'`. From the xPC Target Explorer window **Target RAM size** list, select either `Auto` or `Manual`. If you select `Manual`, enter the amount of RAM, in megabytes, installed on the target PC. This property is set by default to `Auto`. **Target RAM size** defines the total amount of installed RAM in the target PC. This RAM is used for the kernel, target application, data logging, and other functions that use the heap. If **Target RAM size** is set to `Auto`, the target application automatically determines the amount of memory up to 64 MB. If the target PC does not contain more than 64 MB of RAM, or you do not want to use more than 64 MB, select `Auto`. If the target PC has more than 64 MB of RAM, and you want to use more than 64 MB, select `Manual`, and enter the amount of RAM installed in the target PC.

Environment Property	Description
`MaxModelSize`	Values are `'1MB'`, `'4MB'`, and `'16MB'`. From the xPC Target Explorer window **Maximum model size** list, select either `1 MB`, `4 MB`, or `16 MB`. Choosing the maximum model size reserves the specified amount of memory on the target PC for the target application. The remaining memory is used by the kernel and by the heap for data logging. 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.
`HostTargetComm`	Values are `'RS232'` and `'TcpIp'`. From the xPC Target Explorer window **Host target communication** list, select either `RS232` or `TCP/IP`. If you select `RS232`, you also need to set the property `RS232HostPort`. If you select `TCP/IP`, then you also need to set all properties that start with `TcpIp`.
`RS232HostPort`	Values are `'COM1'` and `'COM2'`. From the xPC Target Explorer window **Host port** list, select either `COM1` or `COM2` for the connection on the host computer. xPC Target automatically determines the COM port on the target PC. Before you can select an RS-232 port, you need to set the `HostTargetComm` property to `RS232`.

Environment Property	Description
`RS232Baudrate`	Values are `'115200'`, `'57600'`, `'38400'`, `'19200'`, `'9600'`, `'4800'`, `'2400'`, and `'1200'`. From the **Baud rate** list, select 115200, 57600, 38400, 19200, 9600, 4800, 2400, or 1200.
`TcpIpTargetAddress`	Value is `'xxx.xxx.xxx.xxx'`. In the xPC Target Explorer window **Target PC IP address** box, enter a valid IP address for your target PC. Ask your system administrator for this value. For example, `192.168.0.1`.
`TcpIpTargetPort`	Value is `'xxxxx'`. In the xPC Target Explorer window **TcpIp target port** box, enter a value greater than 20000. This property is set by default to 22222 and should not cause any problems. The number is higher than the reserved area (`telnet`, `ftp`, ...) and it is only of use on the target PC.
`TcpIpSubNetMask`	Value is `'xxx.xxx.xxx.xxx'`. In the xPC Target Explorer window **LAN subnet mask address** text box, enter the subnet mask of your LAN. Ask your system administrator for this value. For example, your subnet mask could be `255.255.255.0`.

Environment Property	Description
TcpIpGateway	Value is 'xxx.xxx.xxx.xxx'. In the xPC Target Explorer window **TcpIp gateway address** box, enter the IP address for your gateway. This property is set by default to 255.255.255.255, which means that a gateway is not used to connect to the target PC. If you communicate with your target PC from within a LAN that uses gateways, and your host and target computers are connected through a gateway, then you need to enter a value for this property. If your LAN does not use gateways, you do not need to change this property. Ask your system administrator.
TcpIpTargetDriver	Values are 'NE2000', 'SMC91C9X', 'I82559', and 'RTLANCE'. From the xPC Target Explorer window **TcpIp target driver** list, select NE2000, SMC91C9X, I82559, or RTLANCE. The Ethernet card provided with xPC Target uses the NE2000 driver.
TcpIpTargetBusType	Values are 'PCI' and 'ISA'. From the xPC Target Explorer window **TcpIp target bus type** list, select either PCI or ISA. This property is set by default to PCI, and determines the bus type of your target PC. You do not need to define a bus type for your host PC, which can be the same or different from the bus type in your target PC. 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
TcpIpTargetISAMemPort	Value is '0xnnnn'. If you are using an ISA bus Ethernet card, you must enter values for the properties TcpIpISAMemPort and TcpIpISAIRQ. The values of these properties must correspond to the jumper settings or ROM settings on your ISA bus Ethernet card. On your ISA bus card, assign an IRQ and I/O port base address by moving the jumpers on the card. Set the I/O port base address to around 0x300. If one of these hardware settings leads to a conflict in your target PC, choose another I/O port base address and make the corresponding changes to your jumper settings.
TcpIpTargetISAIRQ	Value is 'n', where *n* is between 4 and 15. If you are using an ISA bus Ethernet card, you must enter values for the properties TcpIpISAMemPort and TcpIpISAIRQ. The values of these properties must correspond to the jumper settings or ROM settings on the ISA-bus Ethernet card. On your ISA bus card, assign an IRQ and I/O-port base address by moving the jumpers on the card. The MathWorks recommends setting the IRQ to 5, 10, or 11. If one of these hardware settings leads to a conflict in your target PC, choose another IRQ and make the corresponding changes to your jumper settings.

Environment Property	Description
`TargetScope`	Values are `'Disabled'` and `'Enabled'`. From the xPC Target Explorer window **Enable target scope** list, select either `Enabled` or `Disabled`. The property `TargetScope` is set by default to `Enabled`. If you set `TargetScope` to `Disabled`, the target PC displays information as text. To use all the features of the target scope, you also need to install a keyboard and mouse on the target PC.
`TargetMouse`	Values are `'None'`, `'PS2'`, `'RS232 COM1'`, and `'RS232 COM2'`. From the xPC Target Explorer window **Target mouse** list, select `None`, `PS2`, `RS232 COM1`, or `RS232 COM2`. Before you can select a target mouse, you need to set the `Target Scope` property to `Enabled`. **Target mouse** allows you to disable or enable mouse support on the target PC: • If you do not connect a mouse to the target PC, you need to set this property to `None`; otherwise, the target application might not behave properly. • If the target PC supports PS/2 devices (keyboard and mouse) and you connect a PS/2 mouse, set this property to `PS2`. • If you connect a serial RS-232 mouse to the target PC, select either `RS232 COM1` or `RS232 COM2` depending on which serial port you attached the mouse to.

Environment Property	Description
`EmbeddedOption`	Values are `'Disabled'` and `'Enabled'`. This property is read only. Note that the xPC Target Embedded Option is enabled only if you purchase an additional license.
`TargetBoot`	Values are `'BootFloppy'`, `'DOSLoader'`, and `'StandAlone'`. From the xPC Target Explorer window **Target boot mode** list, select `BootFloppy`, `DOSLoader`, or `StandAlone`. If your license file does not include the license for the xPC Target Embedded Option, the **Target boot mode** box is disabled, with `BootFloppy` as your only selection. With the xPC Target Embedded Option licensed and installed, you have the additional choices of `DOSLoader` and `StandAlone`.

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

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

Examples

List the current environment properties.

```
setxpcenv
```

Change the serial communication port of the host PC to `COM2`.

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

See Also

The xPC Target functions `getxpcenv`, `updatexpcenv`, and `xpcbootdisk`. The procedures “Changing Environment Properties with xPC Target Explorer” on page 5-3 and “Changing Environment Properties with a Command-Line Interface” on page 5-6.

Purpose

Start execution of a scope on target PC

Syntax

MATLAB command line

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

start(getscope((target_object, signal_index_vector))
```

Target PC command line

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

Arguments

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

Description

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

Examples

Start one scope with the scope object sc1.

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

or type

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

Start two scopes.

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

or type

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

or type

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

Start all scopes:

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

or type

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

See Also

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

Purpose Start execution of target application on target PC

Syntax

MATLAB command line

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

Target PC command line

```
start
```

Arguments

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

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

Examples Start the target application represented by the target object `tg`.

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

See Also xPC Target target object methods `stop (target object)`, `load`, and `unload`.

Scope object method `stop (scope object)`.

stop (scope object)

Purpose Stop execution of a scope on target PC

Syntax

MATLAB command line

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

```
stop(getscope(target_object, signal_index_vector))
```

Target PC command line

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

Arguments

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

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

Examples Stop one scope represented by the scope object `sc1`.

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

Stop all scopes with a scope object vector `allscopes` created with the command

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

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

or type

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

See Also

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

stop (target object)

Purpose

Stop execution of target application on target PC

Syntax

MATLAB command line

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

Target PC command line

```
stop
```

Arguments

`target_object`	Name of a target object.

Description

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

Examples

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

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

See Also

The xPC Target target object method `start (target object)`. The scope object methods `stop (scope object)` and `start (scope object)`.

Purpose

Test communication between a host and its target computers

Syntax

MATLAB command line

```
targetping(target_object)
target_object.targetping
```

Arguments

`target_object`	Name of a target object.

Description

Method of a target object. Use this method to ping a target PC from the host PC. It returns either `success` or `failed`. If the xPC Target kernel is loaded, running, and communication is working properly, this function returns the value `success`.

This function works with both RS-232 and TCP/IP communication.

Examples

Ping the communication between the host and the target object `tg`.

`targetping(tg)` or `tg.targetping`

See Also

The xPC Target target object methods `delete` and `xpctarget.xpc`.

Purpose Software-trigger start of data acquisition for one or more scopes

Syntax **MATLAB command line**

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

Arguments

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

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

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

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

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

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

plot(sc1.time, sc1.data)

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

Set all scopes to software trigger and trigger to start.

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

Purpose

Remove current target application from target PC

Syntax

MATLAB command line

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

Arguments

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

Description

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

Examples

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

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

See Also

xPC Target methods `load` and `reboot`.

updatexpcenv

Purpose Change current environment properties to new properties

Syntax **MATLAB Command Line**

```
updatexpcenv
```

Description Function to update environment properties. Call the function `updatexpcenv` in the following order:

1 Enter new properties with the function `setxpcenv`.
2 Type `updatexpcenv` to change the current properties to match the new properties.
3 Create a target boot floppy with the function `xpcbootdisk`.

See Also The xPC Target functions `setxpcenv`, `getxpcenv`, and `xpcbootdisk`. The procedures "Changing Environment Properties with xPC Target Explorer" on page 5-3 and "Changing Environment Properties with a Command-Line Interface" on page 5-6.

Purpose Call target object constructor, `xpctarget.xpc`

Note See “xpctarget.xpc” on page 14-116

xpcbootdisk

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

Syntax **MATLAB Command Line**

```
xpcbootdisk
```

Description Function to create an xPC target boot floppy for the current xPC Target environment that has been updated with the function `updatexpcenv`. Creating an xPC Target boot floppy consists of writing the correct bootable kernel image onto the disk. You are asked to insert an empty formatted floppy disk into the 3.5 inch disk drive.

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

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

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

```
xpcbootdisk
```

See Also The xPC Target functions `setxpcenv`, `getxpcenv`, and `updatexpcenv`. The procedures "Changing Environment Properties with xPC Target Explorer" on page 5-3 and "Changing Environment Properties with a Command-Line Interface" on page 5-6.

Purpose Open xPC Target Explorer window

Syntax **MATLAB Command Line**

```
xpcexplr
```

Description This graphical user interface (GUI) allows you to

- Manage an xPC Target system
- Enter and change environment properties
- Create an xPC Target boot disk
- Build, download, and run target applications
- Monitor signals
- Tune parameters

See Also The xPC Target functions `setxpcenv`, `getxpcenv`, `updatexpcenv`, and `xpcbootdisk`. The procedures "Environment Properties for Serial Communication" on page 2-24 and "Environment Properties for Network Communication" on page 2-34.

xpctarget.fs

Purpose

Create xPC Target file system object

Syntax

MATLAB command line

```
filesys_object = xpctarget.fs('mode', 'arg1', 'arg2')
```

Arguments

`filesys_object`	Variable name to reference the file system object.		
`mode`	Optionally, enter the communication mode:		
		`TCPIP`	Specify TCP/IP connection with target PC.
		`RS232`	Specify RS-232 connection with target PC.
`arg1`	Optionally, enter an argument based on the `mode` value:		
		IP address	If `mode` is `'TCPIP'`, enter the IP address of the target PC.
		COM port	If `mode` is `'RS232'`, enter the host COM port.
`arg2`	Optionally, enter an argument based on the `mode` value:		
		Port	If `mode` is `'TCPIP'`, enter the port number for the target PC.
		Baud rate	If `mode` is `'RS232'`, enter the baud rate for the connection between the host and target PC.

Description

Constructor of a file system object. The file system object represents the file system on the target PC. You work with the file system by changing the file system object using methods.

If you have one target PC object, or if you designate a target PC as the default one in your system, use the syntax

```
filesys_object=xpctarget.fs
```

If you have multiple target PCs in your system, or if you want to identify a target PC with the file system object, use the following syntax to create the additional file system objects.

```
filesys_object=xpctarget.fs('mode', 'arg1', 'arg2')
```

Examples

In the following example, a file system object for a target PC with an RS-232 connection is created.

```
fs1=xpctarget.fs('RS232','COM1','115200')

fs1 =
    xpctarget.fs
```

Optionally, if you have an `xpctarget.xpc` object, you can construct an `xpctarget.fs` object by passing the `xpctarget.xpc` object variable to the `xpctarget.fs` constructor as an argument.

```
>> tg1=xpctarget.xpc('RS232','COM1','115200');
>> fs2=xpctarget.fs(tg1)

fs2 =

    xpctarget.fs
```

xpctarget.ftp

Purpose Create xPC Target FTP object

Syntax

MATLAB command line

```
file_object = xpctarget.fs('mode', 'arg1', 'arg2')
```

Arguments

`file_object`	Variable name to reference the FTP object.	
`mode`	Optionally, enter the communication mode:	
	`TCPIP`	Specify TCP/IP connection with target PC.
	`RS232`	Specify RS-232 connection with target PC.
`arg1`	Optionally, enter an argument based on the `mode` value:	
	IP address	If `mode` is `'TCPIP'`, enter the IP address of the target PC.
	COM port	If `mode` is `'RS232'`, enter the host COM port.
`arg2`	Optionally, enter an argument based on the `mode` value:	
	Port	If `mode` is `'TCPIP'`, enter the port number for the target PC.
	Baud rate	If `mode` is `'RS232'`, enter the baud rate for the connection between the host and target PC.

Description Constructor of an FTP object. The FTP object represents the file on the target PC. You work with the file by changing the file object using methods.

If you have one target PC object, or if you designate a target PC as the default one in your system, use the syntax

```
file_object=xpctarget.ftp
```

If you have multiple target PCs in your system, or if you want to identify a target PC 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 PC 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
```

Purpose Create a target object representing target application

Syntax

MATLAB command line

```
target_object = xpctarget.xpc('mode', 'arg1', 'arg2')
```

Arguments

`target_object` Variable name to reference the target object:

`mode` Optionally, enter the communication mode

`TCPIP`	Enable TCP/IP connection with target PC.
`RS232`	Enable RS-232 connection with target PC.

`arg1` Optionally, enter an argument based on the `mode` value:

IP address	If `mode` is `'TCPIP'`, enter the IP address of the target PC.
COM port	If `mode` is `'RS232'`, enter the host COM port.

`arg2` Optionally, enter an argument based on the `mode` value:

Port	If `mode` is `'TCPIP'`, enter the port number for the target PC.
Baud rate	If `mode` is `'RS232'`, enter the baud rate for the connection between the host and target PC.

Description Constructor of a target object. The target object represents the target application and target PC. You make changes to the target application by changing the target object using methods and properties.

If you have one target PC, or if you designate a target PC as the default one in your system, use the syntax

```
target_object=xpctarget.xpc
```

If you have multiple target PCs in your system, use the following syntax to create the additional target objects.

```
target_object=xpctarget.xpc('mode', 'arg1', 'arg2')
```

Examples

Before you build a target application, you can check the connection between your host and target computers by creating a target object, then using the `targetping` method to check the connection.

```
tg = xpctarget.xpc

   xPC Object

       Connected            = Yes
       Application          = loader

tg.targetping

   ans =

   success
```

If you have a second target computer for which you want to check the connection, create a second target object. In the following example, the connection with the second target computer is an RS-232 connection.

```
tg1=xpctarget.xpc('RS232','COM1','115200')

   xPC Object
   Connected            = Yes
   Application          = loader
```

See Also

xPC Target methods `get (target object)`, `set (target object)`, `delete`, and `targetping`.

xpctargetping

Purpose Test communication between host and target computers

Syntax

MATLAB Command Line

```
xpctargetping
```

Examples Check for communication between the host PC and target PC.

```
xpctargetping
```

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

This function works with both RS-232 and TCP/IP communication.

```
ans =
success
```

See Also The xPC Target procedure “Testing and Troubleshooting the Installation” on page 2-45

xpctargetspy

Purpose

Open **Real-Time xPC Target Spy** window on host PC

Syntax

MATLAB Command Line

```
xpctargetspy(target_object)
```

Arguments

`target_object`	Variable name to reference the target object.

Description

This graphical user interface (GUI) allows you to upload displayed data from the target PC. By default, `xpctargetspy` opens a **Real-Time xPC Target Spy** window for the target object, `tg`. If you have multiple target PCs in your system, you can call the `xpctargetspy` function for a particular target object, `target_object`.

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 PC, `tg`, in the MATLAB window, type

```
xpctargetspy
```

To open the **Real-Time xPC Target Spy** window for a target PC, tg1, in the MATLAB window, type

```
xpctargetspy(tg1)
```

xpctest

Purpose

Test the xPC Target installation

Syntax

MATLAB Command Line

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

Arguments

`'noreboot'`	Only one possible option. Skips the reboot test. Use this option if the target hardware does not support software rebooting. Value is `'noreboot'`.

Description

xpctest is a series of xPC Target tests to check the correct functioning of the following xPC Target tasks:

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

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

Examples

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

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

See Also

Procedures "Testing and Troubleshooting the Installation" on page 2-45 and "Test 1, Ping Target System Standard Ping" on page 2-47

Purpose Disconnect target PC from current client application

Syntax

MATLAB Command Line

```
xpcwwwenable
```

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

Index