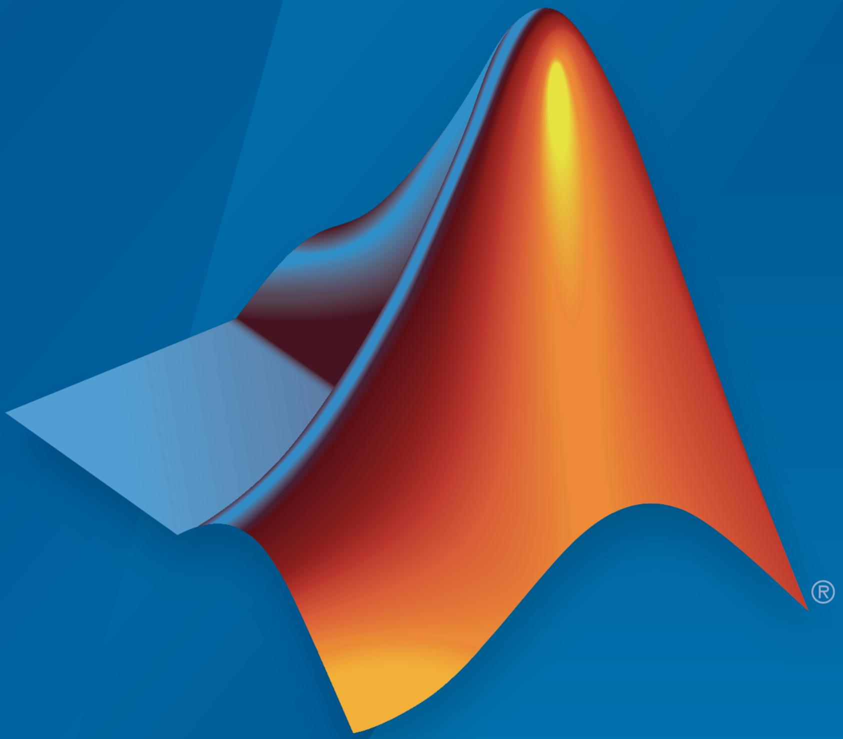
Simulink® Real-Time™

API Guide



MATLAB®&SIMULINK®

R2015a

MathWorks®

How to Contact MathWorks

Latest news:	`www.mathworks.com`
Sales and services:	`www.mathworks.com/sales_and_services`
User community:	`www.mathworks.com/matlabcentral`
Technical support:	`www.mathworks.com/support/contact_us`
Phone:	508-647-7000

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

Simulink® Real-Time™ API Guide

Trademarks

Patents

Revision History

July 2002	Online only	New for Version 2 (Release 13)
October 2002	Online only	Updated for Version 2 (Release 13)
September 2003	Online only	Revised for Version 2.0.1 (Release 13SP1)
June 2004	Online only	Revised for Version 2.5 (Release 14)
August 2004	Online only	Revised for Version 2.6 (Release 14+)
October 2004	Online only	Revised for Version 2.6.1 (Release 14SP1)
November 2004	Online only	Revised for Version 2.7 (Release 14SP1+)
March 2005	Online only	Revised for Version 2.7.2 (Release 14SP2)
September 2005	Online only	Revised for Version 2.8 (Release 14SP3)
March 2006	Online only	Revised for Version 2.9 (Release 2006a)
May 2006	Online only	Revised for Version 3.0 (Release 2006a+)
September 2006	Online only	Revised for Version 3.1 (Release 2006b)
March 2007	Online only	Revised for Version 3.2 (Release 2007a)
September 2007	Online only	Revised for Version 3.3 (Release 2007b)
March 2008	Online only	Revised for Version 3.4 (Release 2008a)
October 2008	Online only	Revised for Version 4.0 (Release 2008b)
March 2009	Online only	Revised for Version 4.1 (Release 2009a)
September 2009	Online only	Revised for Version 4.2 (Release 2009b)
March 2010	Online only	Revised for Version 4.3 (Release 2010a)
September 2010	Online only	Revised for Version 4.4 (Release 2010b)
April 2011	Online only	Revised for Version 5.0 (Release 2011a)
September 2011	Online only	Revised for Version 5.1 (Release 2011b)
March 2012	Online only	Revised for Version 5.2 (Release 2012a)
September 2012	Online only	Revised for Version 5.3 (Release 2012b)
March 2013	Online only	Revised for Version 5.4 (Release 2013a)
September 2013	Online only	Revised for Version 5.5 (Release 2013b)
March 2014	Online only	Revised for Version 6.0 (Release 2014a)
October 2014	Online only	Revised for Version 6.1 (Release 2014b)
March 2015	Online only	Revised for Version 6.2 (Release 2015a)

Contents

1 Introduction

2 Simulink Real-Time API for Microsoft .NET Framework

3 Simulink Real-Time API for C

4 Simulink Real-Time .NET API Examples

5 Simulink Real-Time API Reference for Microsoft .NET Framework

6 Simulink Real-Time API Reference for C

7 MATLAB API

1

Introduction

Simulink Real-Time APIs

The Simulink® Real-Time™ software provides several APIs that enable you to create custom applications to control real-time applications running on target computers. These include Simulink Real-Time MATLAB® Language, the Simulink Real-Time API for Microsoft® .NET Framework, and the Simulink Real-Time C API. These interfaces provide the same functionality for you to write custom solutions (for example, client real-time applications and batch runs) that use the Simulink Real-Time software. The Simulink Real-Time documentation collectively refers to these APIs as Simulink Real-Time API.

The Simulink Real-Time APIs allow you to:

- Establish communication between the development computer and the target computer via an Ethernet or serial link
- Load the real-time application, a `.dlm` file, to the target computer
- Run that application on the target computer
- Monitor the behavior of the real-time application on the target computer
- Stop that application on the target computer
- Unload the real-time application from the target computer
- Close the connection to the target computer

The following sections describe each library:

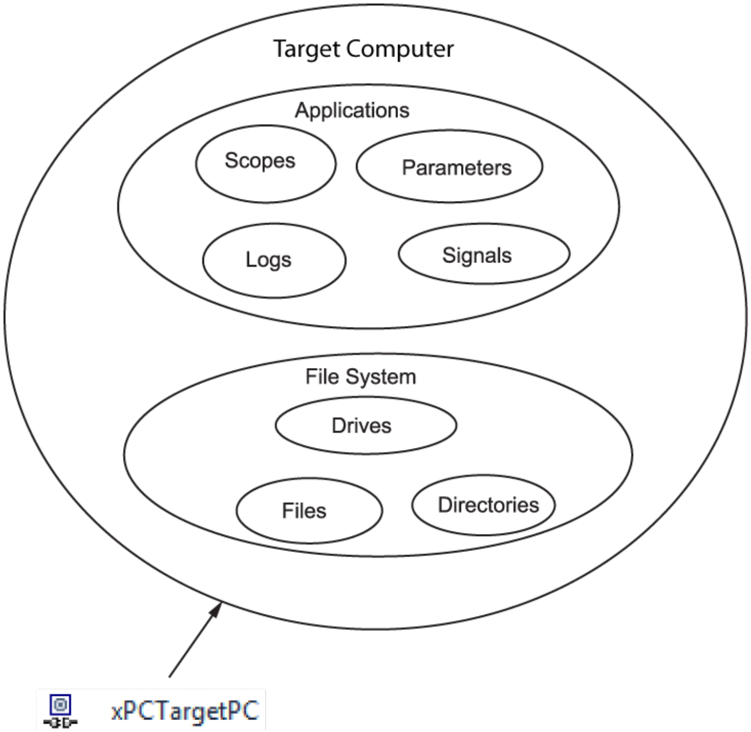
- "Simulink Real-Time API for Microsoft .NET Framework" on page 1-3
- "Simulink Real-Time C API" on page 1-7

Simulink Real-Time API for Microsoft .NET Framework

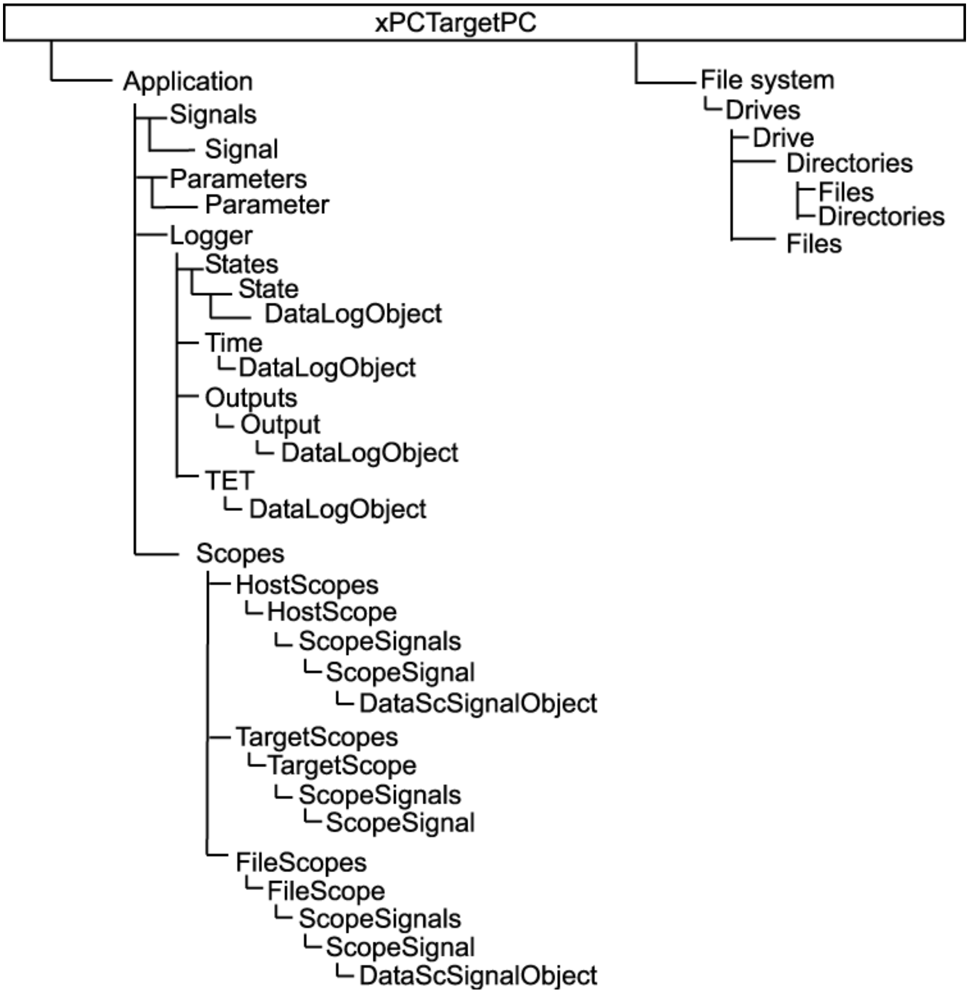
The Simulink Real-Time API for Microsoft .NET Framework consists of objects arranged in hierarchical order. Each of these objects has functions and properties that allow you to manipulate and interact with the API. The API provides a number of object types, including those for the target computer, real-time applications, scopes, and the file system. You can use these API functions from languages and applications that support managed code.

The Microsoft Windows® API supplies the infrastructure for using threads. The Simulink Real-Time API for Microsoft .NET Framework builds on top of that infrastructure to provide a programming model that includes asynchronous support. You do not need prior knowledge of threads programming to use this API.

The Simulink Real-Time .NET object model closely models the Simulink Real-Time system, as shown in the this conceptual diagram.



The API object hierarchy derived from the Simulink Real-Time system is shown in this conceptual diagram.



The key object types are `xPCTargetPC`, `xPCApplication`, and `xPCFileSystem`.

xPCTargetPC Class

The `xPCTargetPC Class` object represents the overall Simulink Real-Time system.

The xPCTargetPC object is at the root level of the object model. After you connect the .NET application running on the development computer to the real-time application running on the target computer, the object exposes information about the Simulink Real-Time session . xPCTargetPC provides many member functions that you use to access information and to manipulate the real-time application and the target computer file system.

An xPCTargetPC object contains two main object types, xPCApplication and xPCFileSystem.

xPCApplication Class

The xPCApplication Class object represents the real-time application that you generate from a Simulink model and download to the target computer.

With the xPCApplication object, you can access application information, change application behavior, and access scope, signal, parameter, and data logging objects:

- xPCScopes Class — Represents a container or placeholder for Simulink Real-Time target, host, and file scopes.
- xPCSignals Class — Represents a container or placeholder for application signals. With this object, you can access one or more xPCSignal objects.
- xPCSignal Class — Represents a specific application signal, which represents the port signal of a nongraphical block output. With this object, you can access signal-related information and monitor signal behavior during simulation.
- xPCParameters Class — Represents a container or placeholder for application parameters. With this object, you can access one or more xPCParameter objects.
- xPCParameter Class — Represents a specific application parameter or a run-time parameter of a specific block. With this object, you can access block parameter information and tune parameter values during simulation.
- xPCAppLogger Class — Represents a placeholder for specific logging objects.

xPCFileSystem

An xPCFileSystem Class object represents the entire Simulink Real-Time file system.

An xPCFileSystem object contains objects like the following:

- `xPCDriveInfo Class` — Represents a volume drive that the target computer recognizes.
- `xPCDirectoryInfo Class` — Represents a target computer folder item.
- `xPCFileInfo Class` — Represents a target computer file item.

Simulink Real-Time C API

The Simulink Real-Time C API consists of a series of C functions that you can call from a C or C++ application. This API is designed for multi-threaded operation. The Simulink Real-Time C API DLL consists of C functions that you can incorporate into a high-level language application. A user can use an application written through either interface to load, run, and monitor a Simulink Real-Time application without interacting with MATLAB. With the Simulink Real-Time C API, you write the application in a high-level language (such as C, C++, or Java®) that works with a Simulink Real-Time application; this option requires that you are an experienced programmer.

The `xpcapi.dll` file contains the Simulink Real-Time C API dynamic link library, which contains over 90 functions you can use to access the real-time application. Because `xpcapi.dll` is a dynamic link library, your program can use run-time linking rather than static linking at compile time. Accessing the Simulink Real-Time C API DLL is beneficial when you are building applications using development environments such as Microsoft Foundation Class Library/Active Template Library (MFC/ATL), DLL, Win32 (non-MFS) program and DLL, and console programs integrating with third-party product APIs (for example, Altia®).

All custom Simulink Real-Time C API applications must link with the `xpcapi.dll` file (Simulink Real-Time C API DLL). Also associated with the dynamic link library is the `xpcinitfree.c` file. This file contains functions that load and unload the Simulink Real-Time C API. You must build this file along with the custom Simulink Real-Time C API application.

The Simulink Real-Time C API consists of blocking functions. For communications between the development and target computers, a default timeout of 5 seconds controls how long a target computer can take to communicate with a development computer.

The documentation reflects the fact that the API is written in the C programming language. However, the API functions are usable from other languages and applications, such as C++ and Java.

Note: To write a non-C application that calls functions in the Simulink Real-Time C API library, refer to the compiler documentation for a description of how to access functions from a library DLL. You must follow these directions to access the Simulink Real-Time C API DLL.

C API Error Messages

The header file *matlabroot*`\toolbox\rtw\targets\xpc\api\xpcapiconst.h` defines these error messages.

Message	Description
ECOMPORTACCFAIL	COM port access failed
ECOMPORTISOPEN	COM port is already opened
ECOMPORTREAD	ReadFile failed while reading from COM port
ECOMPORTWRITE	WriteFile failed while writing to COM port
ECOMTIMEOUT	timeout while receiving: check serial communication
EFILEOPEN	Error opening file
EFILEREAD	Error reading file
EFILERENAME	Error renaming file
EFILEWRITE	Error writing file
EINTERNAL	Internal Error
EINVADDR	Invalid IP Address
EINVARGUMENT	Invalid Argument
EINVALIDMODEL	Model name does not match saved value
EINVBAUDRATE	Invalid value for baudrate
EINVCOMMTYP	Invalid communication type
EINVCOMPORT	COM port can only be 0 or 1 (COM1 or COM2)
EINVDECIMATION	Decimation must be positive
EINVFILENAME	Invalid file name
EINVINSTANDALONE	Command not valid for StandAlone
EINVLGDATA	Invalid lgdata structure
EINVLGINCR	Invalid increment for value equidistant logging
EINVLGMODE	Invalid Logging mode
EINVLOGID	Invalid log identifier
EINVNUMPARAMS	Invalid number of parameters

Message	Description
EINVNUMSIGNALS	Invalid number of signals
EINVPARIDX	Invalid parameter index
EINVPORT	Invalid Port Number
EINVSCIDX	Invalid Scope Index
EINVSCTYPE	Invalid Scope type
EINVSIGIDX	Invalid Signal index
EINVTRIGMODE	Invalid trigger mode
EINVTRIGSLOPE	Invalid Trigger Slope Value
EINVTRSCIDX	Invalid Trigger Scope index
EINVNUMSAMP	Number of samples must be nonnegative
EINVSTARTVAL	Invalid value for "start"
EINVTFIN	Invalid value for TFinal
EINVTS	Invalid value for Ts (must be between 8e-6 and 10)
EINVWSVER	Invalid Winsock version (1.1 needed)
EINVXPCVERSION	Target has an invalid version of Simulink Real-Time
ELOADAPPFIRST	Load the application first
ELOGGINGDISABLED	Logging is disabled
EMALFORMED	Malformed message
EMEMALLOC	Memory allocation error
ENODATALOGGED	No data has been logged
ENOERR	No error
ENOFREEPORT	No free Port in C API
ENOMORECHANNELS	No more channels in scope
ENOSPACE	Space not allocated
EOUTPUTLOGDISABLED	Output Logging is disabled
EPARNOTFOUND	Parameter not found
EPARSIZMISMATCH	Parameter Size mismatch

Message	Description
EPINGCONNECT	Could not connect to Ping socket
EPINGPORTOPEN	Error opening Ping port
EPINGSOCKET	Ping socket error
EPORTCLOSED	Port is not open
ERUNSIMFIRST	Run simulation first
ESCFINVALIDFNAME	Invalid filename tag used for dynamic file name
ESCFISNOTAUTO	Autorestart must be enabled for dynamic file names
ESCFNUMISNOTMULT	MaxWriteFileSize must be a multiple of the writesize
ESCTYPENOTTGT	Scope Type is not "Target"
ESIGLABELNOTFOUND	Signal label not found
ESIGLABELNOTUNIQUE	Ambiguous signal label (signal labels are not unique)
ESIGNOTFOUND	Signal not found
ESOCKOPEN	Socket Open Error
ESTARTSIMFIRST	Start simulation first
ESTATELOGDISABLED	State Logging is disabled
ESTOPSCFIRST	Stop scope first
ESTOPSIMFIRST	Stop simulation first
ETCPCONNECT	TCP/IP Connect Error
ETCPREAD	TCP/IP Read Error
ETCPTIMEOUT	TCP/IP timeout while receiving data
ETCPWRITE	TCP/IP Write error
ETETLOGDISABLED	TET Logging is disabled
ETGTMEMALLOC	Target memory allocation failed
ETIMELOGDISABLED	Time Logging is disabled
ETOOMANYSAMPLES	Too Many Samples requested
ETOOMANYSCOPES	Too many scopes are present

Message	Description
ETOOMANYSIGNALS	Too many signals in Scope
EUNLOADAPPFIRST	Unload the application first
EUSEDYNSCOPE	Use DYNAMIC_SCOPE flag at compile time
EWRITEFILE	LoadDLM: WriteFile Error
EWSINIT	WINSOCK: Initialization Error
EWSNOTREADY	Winsock not ready

2

Simulink Real-Time API for Microsoft .NET Framework

Using the Simulink Real-Time API for Microsoft .NET Framework

The Simulink Real-Time API for Microsoft .NET Framework is a fully managed and fully programmable .NET framework component. It contains components and types that enable you to quickly design Simulink Real-Time client applications. Although the framework is designed to work with Microsoft Visual Studio®, you can use it with other development environments and programming languages that support the .NET framework.

The Simulink Real-Time .NET API includes the following features.

- Microsoft Visual Studio design time.
- Intuitive object model (modeled after the Simulink Real-Time system environment).
- Simplified client model programming for asynchronous communication with the target computer.

The Simulink Real-Time .NET API provides multiple ways for you to interface client-side applications with target computers, including outside the MATLAB environment. For example:

- Visual instrumentation for your real-time application.
- Custom applications to perform data observation, collection, and archiving.
- Real-time application debugging from a remote client computer.
- Calibration, test, and evaluation of real-time processes.
- Real-time data analysis.
- Batch processing and automation scripts, which can run in a shell environment (such as PowerShell™) or as a process console standalone application (`.exe` file).

The Simulink Real-Time API for .NET framework supports a run-time user-driven mode of execution and an optional developer-driven mode of execution, or design-time capability. You can integrate the design-time capability with the Microsoft Visual Studio IDE. The following operations are available:

- Drag-and-drop UI elements into the form design
- Configure properties using a design-time properties window
- Delete UI elements from the form design

For more information on using Microsoft Visual Studio .NET, see http://msdn.microsoft.com/en-us/library/aa973739(v=vs.71).aspx.

For some examples of .NET applications, see “Simulink Real-Time .NET API Client Application Examples”

Simulink Real-Time .NET API Application Creation

Before creating your Microsoft .NET Framework client application, you must set up the development environment. In addition to the products listed in System Requirements , install the following products:

- Third-Party Development Environment — To build a custom application that references interfaces in the Simulink Real-Time API for the .NET Framework, use a third-party development environment and compiler that can interact with .NET. For example, the Windows PowerShell, Microsoft Visual Studio, and the MATLAB environments.
- Third-Party Compiler — To build a custom application (`.exe`, `DLL`) that calls functions from the Simulink Real-Time API libraries, use a third-party compiler that generates code for Win32 systems. You can write client applications that call these functions in another high-level language, such as C#, C++, or C.

In the Visual Studio environment, use the `xPCTargetPC` component:

1. Add the `xPCTargetPC` component to the Visual Studio Toolbox.
2. To use this component, create a Windows application.
3. Add an `xPCTargetPC` object to the application form by dragging an `xPCTargetPC` control from the Toolbox window to the design surface.
4. To explore and customize the `xPCTargetPC` properties, click the `xPCTargetPC` control in the design surface.

   The Visual Studio **Properties** window opens. In the **Properties** window, the `xPCTargetPC` control makes available its data and appearance properties.
5. Add a reference for `xPCFramework.dll` to your project by including the following in your code.

   ```
   using MathWorks.xPCTarget.FrameWork;
   ```

   You can then access the types available from the Simulink Real-Time environment, for example, when creating a console application.
6. To use the design-time capability of the Microsoft Visual Studio environment, copy the `xpcapi.dll` file to the same folder as the application executable. The application also needs this file to execute.

   Simulink Real-Time includes a 32-bit and a 64-bit version of the `xpcapi.dll`. If the target computer is a 64-bit computer, you must use the 64-bit version.

- On 64-bit platforms, if you build a 64-bit real-time application in the Microsoft Visual Studio environment, and want to use the `xPCTargetPC` nonvisual component, do the following.

  - Place the 32-bit version of `xpcapi.dll` in the solution folder.
  - Place the 64-bit version of `xpcapi.dll` in the application folder that contains the `.exe` file.

  Placing the 32-bit version of `xpcapi.dll` in the solution folder enables you to use the design time capabilities of the Visual Studio environment.
- You can connect a target computer to only one development computer at a time. Before starting your .NET client application, be sure to disconnect the target computer from the development computer (`xPCTargetPC.disconnect`). You can use the `slrtpingtarget` from the Command Window to verify that the computers are not connected. When execution is finished, this function disconnects from the target computer.

  If your development computer has additional network resources, you can connect additional target computers to the same development computer.
- When your .NET client application starts, first connect the development computer to the target computer (`xPCTargetPC.connect`), and then test the link between the development and target computers (`xPCTargetPC.ping`).

Simulink Real-Time .NET API Application Deployment

To deploy your Microsoft .NET Framework client application, such as a UI:

- You must have a Simulink Real-Time license to deploy or distribute your client application.
- When you build your application, the Visual Studio software builds the application files for your executable, including a `*.exe` file. When you deploy or distribute your application, include these files in the same folder.
- Keep in mind the relationship between the client application, `xPCFramework.dll`, and `xpcapi.dll`. In particular, the application depends on `xPCFramework.dll`, which depends on `pcapi.dll`.

  Be sure to provide the version of `xpcapi.dll` (32-bit or 64-bit) for which your application is built.

Simulink Real-Time .NET API Client Application Examples

Simulink Real-Time includes examples showing how to the Simulink Real-Time API for Microsoft .NET Framework to create client applications that run on the development computer and interface with a model downloaded on the target computer.

The example Simple Client Application with the .NET API shows two client applications, `Example 1` and `Example 2`.

- `Example 1` — Provides a UI with buttons, text boxes, and a track bar through which you can enter the IP address port of the target computer with which you want to connect.
- `Example 2` — Provides a UI similar to that in `Example 1`, with also a chart that displays signals from the `xpcosc` real-time application.

Another example, `FileSystemBrowse`, provides a file browser that runs on the development computer and connects to the target computer to browse its file system.

`FileSystemBrowse` is located in:

```
matlabroot\toolbox\rtw\targets\xpc\api\xPCFrameworkSamples\FileSystemBrowse
```

`FileSystemBrowse` is a C# project developed with the Microsoft Visual Studio 2008 IDE. See the `Readme.txt` file in the example folder for instructions on how to access and build the example code.

Simulink Real-Time API for C

Using the C API

Keep the following guidelines in mind when you begin to write Simulink Real-Time C API applications with the Simulink Real-Time C API DLL:

- Carefully match the function data types as documented in the function reference. For C, the API includes a header file that matches the data types.
- To write a non-C application that calls functions in the Simulink Real-Time C API library, refer to the compiler documentation for a description of how to access functions from a library DLL. You must follow these directions to access the Simulink Real-Time C API DLL
- You can work with Simulink Real-Time applications with either MATLAB or a Simulink Real-Time C API application. If you are working with a Simulink Real-Time application simultaneously with a MATLAB session interacting with the target, keep in mind that only one application can access the target computer at a time. To move from the MATLAB session to your application, in the MATLAB Command Window, type

  ```
  close(slrt)
  ```

  This frees the connection to the target computer for use by your Simulink Real-Time C API application. Conversely, you will need to quit your application, or do the equivalent of calling the function `xPCClosePort`, to access the target from a MATLAB session.
- The Simulink Real-Time C API functions that communicate with the target computer check for timeouts during communication. If a timeout occurs, these functions will exit with the global variable `xPCError` set to either `ECOMTIMEOUT` (serial connections) or `ETCPTIMEOUT` (TCP/IP connections). Use the `xPCGetLoadTimeOut` and `xPCSetLoadTimeOut` functions to get and set the timeout values, respectively.

There are a few things that are not covered in the reference topics for the individual functions, because they are common to almost all the functions in the Simulink Real-Time C API. These are

- Almost every function (except `xPCOpenSerialPort`, `xPCOpenTcpIpPort`, `xPCGetLastError`, and `xPCErrorMsg`) has as one of its parameters the integer variable *`port`*. This variable is returned by `xPCOpenSerialPort` and `xPCOpenTcpIpPort`, and should be used to represent the communications link with the target computer.

- Almost every function (except xPCGetLastError and xPCErrorMsg) sets a global error value in case of error. The application obtains this value by calling the function xPCGetLastError, and retrieves a descriptive string about the error by using the function xPCErrorMsg. Although the actual error values are subject to change, a zero value typically means that the operation completed without producing an error, while a nonzero value typically signifies an error condition. Note also that the library resets the error value every time an API function is called; therefore, your application should check the error status as soon as possible after a function call.

  Some functions also use their return values (if applicable) to signify that an error has occurred. In these cases as well, you can obtain the exact error with xPCGetLastError.

Simulink Real-Time .NET API Examples

Visual Basic GUI Using .NET

To help you better understand and quickly begin to use .NET API functions to create custom GUI applications, the Simulink Real-Time environment provides a number of API examples and scripts in the *`matlabroot`*`\toolbox\rtw\targets\xpc\api` folder. This topic briefly describes those examples and scripts.

The Microsoft Visual Basic .NET example illustrates how to create a custom GUI that connects to a target computer with a downloaded real-time application. The solution file for this example is located in

`matlabroot``\toolbox\rtw\targets\xpc\api\VBNET\SigsAndParamsDemo`

- `bin` — Contains the executable for the `Demo` project and the `xpcapi.dll` file
- `Demo.sln` — Contains a solution file for the `Demo` project

The `Demo.sln` file contains the Visual Basic .NET files required to run the windows form application. This example is a functional application that you can use as a template to create your own custom GUIs.

In this section...

Before Starting

To use the `Demo` solution, you need

- A target computer running a current Simulink Real-Time kernel
- A development computer running the MATLAB software interface, connected to the target computer via RS-232 or TCP/IP
- A real-time application loaded on the target computer

The Simulink Real-Time product ships with an executable version of the example. If you want to rebuild the `Demo` solution, of if you want to write your own custom GUIs like this one, you need Microsoft Visual Basic .NET installed on the development computer.

Note: The Simulink Real-Time software allows you to create applications, such as GUIs, to interact with a target computer with .NET API functions. "Visual Basic GUI Using .NET" on page 4-2 describes this in detail.

Accessing the Demo Project Solution

To access the `Demo` solution,

1. Copy the contents of the `VBNET` folder to a writable folder of your choice.
2. Change folder to the one that contains your copy of the `Demo` solution.
3. Double-click `demo.sln`.

   The Microsoft Development Environment for Visual Basic application starts.
4. In the **Solution Explorer** pane, double-click `Form1.vb` to display the `Demo` solution form.

   The form is displayed. You can inspect the layout of the example.
5. To inspect the form code, select the **View** menu `Code` option.

   The Visual Basic code for the form is displayed.

Rebuilding the Demo Project Solution

To rebuild the `Demo` solution,

1. Double-click `demo.sln`.

   The Microsoft Development Environment for Visual Basic application starts.
2. Select the **Build** menu `Build Solution` option.

Using the Demo Executable

To use the `Demo` solution executable,

1. Change folder to the one that contains your copy of the `Demo` solution.
2. Change folder to the `bin` folder.
3. Double-click `Demo1.exe`.

The GUI is displayed.

5

Simulink Real-Time API Reference for Microsoft .NET Framework

xPCFileScopeCollection.Add

Create `xPCFileScope` object with next available scope ID as key

Syntax

```
public xPCFileScope Add()
public xPCFileScope Add(int ID)
public IList<xPCFileScope> Add(int[] arrayOfIDs)
IList
```

Description

Class: `xPCFileScopeCollection Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public xPCFileScope Add()` creates xPCFileScope object with the next available scope ID as key. It then adds xPCFileScope object to xPCFileScopeCollection object.

`public xPCFileScope Add(int ID)` creates xPCFileScope object with *`ID`* as key. *`ID`* is 32-bit integer that specifies an ID for the scope object.

`public IList<xPCFileScope> Add(int[] arrayOfIDs)` creates an `IList` of xPCFileScope objects with an array of IDs as keys. *`arrayOfIDs`* is an array of 32-bit integers that specifies an array of IDs for scope objects.

xPCFileScopeSignalCollection.Add

Add signals to file scope

Syntax

```
public xPCFileScopeSignal Add(xPCSignal signal)
public xPCFileScopeSignal Add(string blkPath)
public xPCFileScopeSignal Add(int sigId)
public IList<xPCFileScopeSignal> Add(int[] sigIds)
```

Description

Class: `xPCFileScopeSignalCollection Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public xPCFileScopeSignal Add(xPCSignal signal)` adds signals to the file scope. It creates an xPCFileScopeSignal object with ***`signal`***. ***`signal`*** is the xPCSignal object that represents the actual signal. This method returns a file scope signal object of type xPCFileScopeSignal.

`public xPCFileScopeSignal Add(string blkPath)` adds signal to the file scope. It creates an xPCFileScopeSignal object that ***`blkPath`*** specifies. ***`blkPath`*** is a string that specifies the signal name (block path). This method returns a file scope signal object of type xPCFileScopeSignal.

`public xPCFileScopeSignal Add(int sigId)` adds signals to the file scope. It creates an xPCFileScopeSignal object specified with ***`sigId`***. ***`sigId`*** is a 32-bit integer that represents the actual signal. This method returns a file scope signal object of type xPCFileScopeSignal.

`public IList<xPCFileScopeSignal> Add(int[] sigIds)` adds signals to the file scope. It creates an IList of xPCFileScopeSignal objects, one for each signal in the array of IDs. *`sigIds`* is an array of 32-bit integers that specifies an array of IDs that represent the actual signals. This method returns an ILIST of xPCFileScopeSignal objects.

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCHostScopeCollection.Add

Create `xPCHostScope` object with next available scope ID as key

Syntax

```
public xPCHostScope Add()
public xPCHostScope Add(int ID)
public IList<xPCHostScope> Add(int[] arrayOfIDs)
```

Description

Class: `xPCHostScopeCollection Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public xPCHostScope Add()` creates xPCHostScope object with the next available scope ID as key. It then adds an xPCHostScope object to xPCHostScopeCollection object. This method returns an xPCHostScopeObject object.

`public xPCHostScope Add(int ID)` creates xPCHostScope object with *`ID`* as key. *`ID`* is 32-bit integer that specifies an ID for the scope object. This method returns an xPCHostScopeObject object.

`public IList<xPCHostScope> Add(int[] arrayOfIDs)` creates an ILIST of xPCHostScope objects with an array of IDs as keys. *`arrayOfIDs`* is an array of 32-bit integers that specifies an array of IDs for scope objects.

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCHostScopeSignalCollection.Add

Add signals to host scope

Syntax

```
public xPCHostScopeSignal Add(xPCSignal signal)
public xPCHostScopeSignal Add(string blkpath)
public xPCHostScopeSignal Add(int sigId)
public IList<xPCHostScopeSignal> Add(int[] sigIds)
```

Description

Class: `xPCHostScopeSignalCollection Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public xPCHostScopeSignal Add(xPCSignal signal)` adds signals to the host scope. It creates xPCHostScopeSignal object with *`signal`*. *`signal`* is the xPCSignal object that represents the actual signal. This method returns an xPCHostScopeSignal object.

`public xPCHostScopeSignal Add(string blkpath)` adds signal to the host scope. It creates an xPCHostScopeSignal object that *`blkPath`* specifies. *`blkPath`* is a string that specifies the signal name (block path). This method returns a host scope signal object of type xPCHostScopeSignal.

`public xPCHostScopeSignal Add(int sigId)` adds signals to the host scope. It creates an xPCHostScopeSignal object specified with *`sigId`*. *`sigId`* is a 32-bit integer that represents the actual signal. This method returns a host scope signal object of type xPCHostScopeSignal.

`public IList<xPCHostScopeSignal> Add(int[] sigIds)` adds signals to the host scope. It creates an ILIST of xPCHostScopeSignal objects, one for each signal in the array of IDs. *`sigIds`* is an array of 32-bit integers that specifies an array of IDs that represent the actual signals. This method returns an ILIST of xPCHostScopeSignal objects.

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCTargetScopeCollection.Add

Create `xPCTargetScope` object

Syntax

```
public xPCTargetScope Add()
public xPCTargetScope Add(int ID)
public IList<xPCTargetScope> Add(int[] arrayOfIDs)
```

Description

Class: `xPCTargetScopeCollection Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public xPCTargetScope Add()` creates xPCTargetScope object with the next available scope ID as key. It then adds xPCTargetScope object to xPCTargetScopeCollection object. This method returns an xPCTargetScope object.

`public xPCTargetScope Add(int ID)` creates xPCTargetScope object with *`ID`* as key. *`ID`* is 32-bit integer that specifies an ID for the scope object. This method returns an xPCTargetScope object.

`public IList<xPCTargetScope> Add(int[] arrayOfIDs)` creates an ILIST of xPCTargetScope objects with an array of IDs as keys. *`arrayOfIDs`* is an array of 32-bit integers that specifies an array of IDs for scope objects. This method returns an ILIST of xPCTargetScope objects.

xPCTargetScopeSignalCollection.Add

Create `xPCTargetScopeSignal` object

Syntax

```
public xPCTgtScopeSignal Add(xPCSignal signal)
public xPCTgtScopeSignal Add(string blkPath)
public xPCTgtScopeSignal Add(int sigId)
public IList<xPCTgtScopeSignal> Add(int[] sigIds)
```

Description

Class: `xPCTargetScopeSignalCollection Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public xPCTgtScopeSignal Add(xPCSignal signal)` creates xPCTargetScopeSignal object with *`signal`*. It then adds xPCTargetScopeSignal object to xPCTargetScopeSignalCollection object. *`signal`* is of type xPCSignal. This method returns an xPCTargetScopeSignal object.

`public xPCTgtScopeSignal Add(string blkPath)` adds signal to the target scope. It creates an xPCTargetScopeSignal object that *`blkPath`* specifies. *`blkPath`* is a string that specifies the signal name (block path). This method returns a target scope signal object of type xPCTgtScopeSignal.

`public xPCTgtScopeSignal Add(int sigId)` creates xPCTargetScopeSignal object with *`sigId`*. It then adds xPCTargetScopeSignal object to xPCTargetScopeSignalCollection object. *`sigId`* is a 32-bit integer. This method returns an xPCTargetScopeSignal object.

`public IList<xPCTgtScopeSignal> Add(int[] sigIds)` creates an ILIST of xPCTargetScopeSignal objects with an array of IDs. *`sigIds`* is an array of 32-bit integers that specifies an array of IDs for file scope signal objects.

Exception

Exception	Condition
xPCException	When problem occurs, query xPCException object Reason property.

xPCFileStream.Close

Close current stream

Syntax

```
public void Close()
```

Description

Class: `xPCFileStream Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void Close()` close the current stream and releases the resources (such as file handles) associated with it.

Exception

Exception	Condition
`xPCException`	When problem occurs, query xPCException object `Reason` property.

xPCTargetPC.Connect

Establish connection to target computer

Syntax

```
public void Connect()
```

Description

Class: `xPCTargetPC Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void Connect()` establishes a connection to a remote target computer.

Exception

Exception	Condition
`xPCException`	When problem occurs, query xPCException object `Reason` property.

xPCTargetPC.ConnectAsync

Asynchronous request for target computer connection

Syntax

```
public void ConnectAsync()
```

Description

Class: `xPCTargetPC Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void ConnectAsync()` begins an asynchronous request for a target computer connection.

Exception

Exception	Condition
`InvalidOperation-Exception`	When another thread uses this method.

xPCTargetPC.ConnectCompleted

Event when `xPCTargetPC.ConnectAsync` is complete

Syntax

```
public event ConnectCompleted ConnectCompleted
```

Description

Class: `xPCTargetPC Class`

Event

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public event ConnectCompleted ConnectCompleted` occurs when an asynchronous connect operation is complete.

xPCTargetPC.Connected

Event after `xPCTargetPC.Connect` is complete

Syntax

```
public event EventHandler Connected
```

Description

Class: `xPCTargetPC Class`

Event

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public event EventHandler Connected` occurs after a connect operation is complete.

xPCTargetPC.Connecting

Event before `xPCTargetPC.Connect` starts

Syntax

```
public event EventHandler Connecting
```

Description

Class: `xPCTargetPC Class`

Event

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public event EventHandler Connecting` occurs before connect operation starts.

xPCFileInfo.CopyToHost

Copy file from target computer file system to development computer file system

Syntax

```
public FileInfo CopyToHost(string HostDestFileName)
```

Description

Class: `xPCFileInfo Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public FileInfo CopyToHost(string HostDestFileName)` copies file, *`HostDestFileName`*, from target computer file system to new location on development computer file system. *`HostDestFileName`* is a string that specifies the full path name for the file.

Exception

Exception	Condition
`ArgumentException`	*`HostDestFileName`* is empty, contains only white spaces, or contains invalid characters.
`ArgumentNullException`	`HostDestFileName` is NULL reference.
`NotSupportedException`	*`HostDestFileName`* contains a colon (:) in the middle of the string.
`PathTooLongException`	The specified path, file name, or both in *`HostDestFileName`* exceed the system-defined maximum length. For example, on Windows platforms, path names must be less than 248 characters. File names must be less than 260 characters.

Exception	Condition
`SecurityException`	Caller does not have required permission.
`UnauthorizedAccessException`	System does not allow access to *`HostDestFileName`*.
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCFileInfo.Create

Create file in specified path

Syntax

```
public xPCFileStream Create()
```

Description

Class: `xPCFileInfo Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public xPCFileStream Create()` create file in specified path.

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCFileSystem.Create

Create folder

Syntax

```
public xPCDirectoryInfo CreateDirectory(string path)
```

Description

Class: `xPCFileSystem Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public xPCDirectoryInfo CreateDirectory(string path)` creates folder on the target computer file system. *`path`* is a string that specifies the full path name for the new folder. This method returns an xPCDirectoryInfo object.

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCDirectoryInfo.Create

Create folder

Syntax

```
public void Create()
```

Description

Class: `xPCDirectoryInfo Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void Create()` creates a folder.

xPCFileSystemInfo.Delete

Delete current file or folder

Syntax

```
public abstract void Delete()
```

Description

Class: `xPCFileSystemInfo Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public abstract void Delete()` deletes the current file or folder on the target computer file system.

xPCDirectoryInfo.Delete

Delete empty `xPCDirectoryInfo` object

Syntax

```
public override void Delete()
```

Description

Class: `xPCDirectoryInfo Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public override void Delete()` deletes an empty xPCDirectoryInfo object.

xPCFileInfo.Delete

Permanently delete file on target computer

Syntax

```
public override void Delete()
```

Description

Class: `xPCFileInfo Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public override void Delete()` permanently deletes files from the target computer.

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCTargetPC.Disconnect

Disconnect from target computer

Syntax

```
public void Disconnect()
```

Description

Class: `xPCTargetPC Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void Disconnect()` closes the connection to the target computer.

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCTargetPC.DisconnectAsync

Asynchronous request to disconnect from target computer

Syntax

```
public void DisconnectAsync()
```

Description

Class: `xPCTargetPC Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void DisconnectAsync()` begins an asynchronous request to disconnect from the target computer.

Exception

Exception	Condition
`InvalidOperation-Exception`	When another thread uses this method.

xPCTargetPC.DisconnectCompleted

Event when `xPCTargetPC.DisconnectAsync` is complete

Syntax

```
public event DisconnectCompletedEventHandler DisconnectCompleted
```

Description

Class: `xPCTargetPC Class`

Event

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public event DisconnectCompletedEventHandler DisconnectCompleted` occurs when an asynchronous disconnect operation is complete.

xPCTargetPC.Disconnected

Event after `xPCTargetPC.Disconnect` is complete

Syntax

```
public event EventHandler Disconnected
```

Description

Class: `xPCTargetPC Class`

Event

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public event EventHandler Disconnected` occurs after a disconnect operation is complete.

xPCTargetPC.Disconnecting

Event before `xPCTargetPC.Disconnect` starts

Syntax

```
public event EventHandler Disconnecting
```

Description

Class: `xPCTargetPC Class`

Event

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public event EventHandler Disconnecting` occurs before a disconnect operation starts.

xPCTargetPC.Dispose

Clean up used resources

Syntax

```
public void Dispose()
```

Description

Class: `xPCTargetPC Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void Dispose()` cleans up used resources.

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCTargetPC.Disposed

Event after `xPCTargetPC.Dispose` is complete

Syntax

```
public event EventHandler Disposed
```

Description

Class: `xPCTargetPC Class`

Event

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public event EventHandler Disposed` occurs after the disposal of used resources is complete.

xPCFileSystem.GetCurrentDirectory

Current working folder for real-time application

Syntax

```
public string GetCurrentDirectory()
```

Description

Class: `xPCFileSystem Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public string GetCurrentDirectory()` gets the current working folder of the real-time application. This method returns the current working folder name as a string.

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCDataLoggingObject.GetData

Copy signal data from target computer

Syntax

```
public double[] GetData()
```

Description

Class: `xPCDataLoggingObject Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public double[] GetData()` copies logged data from the target computer to the development computer.

xPCDataFileScSignalObject.GetData

Copy file scope signal data from target computer

Syntax

```
public double[] GetData()
```

Description

Class: `xPCDataFileScSignalObject Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public double[] GetData()` copies logged file scope signal data from the target computer to the development computer.

xPCDataHostScSignalObject.GetData

Copy host scope signal data from target computer

Syntax

```
public double[] GetData()
```

Description

Class: `xPCDataHostScSignalObject Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public double[] GetData()` copies logged host scope signal data from the target computer to the development computer.

xPCDataLoggingObject.GetDataAsync

Asynchronously copy signal data from target computer

Syntax

```
public void GetDataAsync()
public void GetDataAsync(Object taskId)
```

Description

Class: `xPCDataLoggingObject Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void GetDataAsync()` asynchronously copies the logged data from the target computer without blocking the calling thread.

`public void GetDataAsync(Object taskId)` receives *`taskId`* (user-defined object) when the method copies the logged data.

xPCDataFileScSignalObject.GetDataAsync

Asynchronously copy file scope signal data from target computer

Syntax

```
public void GetDataAsync()
public void GetDataAsync(Object taskId)
```

Description

Class: `xPCDataFileScSignalObject Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void GetDataAsync()` asynchronously copies the file scope signal logged data from the target computer without blocking the calling thread.

`public void GetDataAsync(Object taskId)` receives *`taskId`* (user-defined object) when the method copies the file scope signal logged data. In other words, when the asynchronous operation is complete.

Exception

Exception	Condition
`InvalidOperation-Exception`	When another thread uses this method.

xPCDataHostScSignalObject.GetDataAsync

Asynchronously copy host scope signal data from target computer

Syntax

```
public void GetDataAsync()
public void GetDataAsync(Object taskId)
```

Description

Class: `xPCDataHostScSignalObject Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void GetDataAsync()` asynchronously copies the host scope signal logged data from the target computer without blocking the calling thread.

`public void GetDataAsync(Object taskId)` receives *`taskId`* (user-defined object) when the method copies the host scope signal logged data. In other words, when the asynchronous operation is complete.

Exception

Exception	Condition
`InvalidOperation-Exception`	When another thread uses this method.

xPCDataLoggingObject.GetDataCompleted

Event when `xPCDataLoggingObject.GetDataAsync` is complete

Syntax

```
public event GetDataCompletedEventHandler GetDataCompleted
```

Description

Class: `xPCDataLoggingObject Class`

Event

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public event GetDataCompletedEventHandler GetDataCompleted` occurs when the asynchronous copying of logged data is complete.

xPCDataFileScSignalObject.GetDataCompleted

Event when `xPCDataFileScSignalObject.GetDataAsync` is complete

Syntax

```
public event GetFileScSignalDataCompletedEventHandler
GetDataCompleted
```

Description

Class: `xPCDataFileScSignalObject Class`

Event

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public event GetFileScSignalDataCompletedEventHandler GetDataCompleted` occurs when the asynchronous copying of file scope signal logged data is complete.

xPCDataHostScSignalObject.GetDataCompleted

Event when `xPCDataHostScSignalObject.GetDataAsync` is complete

Syntax

```
public event GetDataCompletedEventHandler GetDataCompleted
```

Description

Class: `xPCDataHostScSignalObject Class`

Event

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public event GetDataCompletedEventHandler GetDataCompleted` occurs when the asynchronous copying of host scope signal logged data is complete.

xPCDirectoryInfo.GetDirectories

Subfolders of current folder

Syntax

```
public xPCDirectoryInfo[] GetDirectories()
```

Description

Class: `xPCDirectoryInfo Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public xPCDirectoryInfo[] GetDirectories()` returns the subfolders of the current folder. This method returns the list of subfolders as an xPCDirectoryInfo array.

xPCFileSystem.GetDrives

Drive names for logical drives on target computer

Syntax

```
public xPCDriveInfo[] GetDrives()
```

Description

Class: `xPCFileSystem Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public xPCDriveInfo[] GetDrives()` retrieves the drive names of the logical drives on the target computer. This method returns an xPCDriveInfo array.

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCDirectoryInfo.GetFiles

File list from current folder

Syntax

```
public xPCFileInfo[] GetFiles()
```

Description

Class: `xPCDirectoryInfo Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public xPCFileInfo[] GetFiles()` returns a file list from the current folder. This method returns the list of files as an xPCFileInfo array.

xPCDirectoryInfo.GetFileSystemInfos

File system information for files and subfolders in folder

Syntax

```
public xPCFileSystemInfo[] GetFileSystemInfos()
```

Description

Class: `xPCDirectoryInfo Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public xPCFileSystemInfo[] GetFileSystemInfos()` returns an array of strongly typed xPCFileSystemInfo entries. These entries represent the files and subfolders in a folder.

xPCParameter.GetParam

Get parameter values from target computer

Syntax

```
public double[] GetParam()
```

Description

Class: `xPCParameter Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public double[] GetParam()` gets parameter values from the target computer as an array of doubles.

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCParameter.GetParamAsync

Asynchronous request to get parameter values from target computer

Syntax

```
public void GetParamAsync()
public void GetParamAsync(Object taskId)
```

Description

Class: `xPCParameter Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void GetParamAsync()` begins an asynchronous request to get parameter values from the target computer. This method does not block the calling thread.

`public void GetParamAsync(Object taskId)` receives a user-defined object when it completes its asynchronous request. *`taskId`* is a user-defined object that you can have passed to the `GetParamAsync` method upon completion.

Exception

Exception	Condition
`InvalidOperation-Exception`	When another thread uses this method.

xPCParameter.GetParamCompleted

Event when `xPCParameter.GetParamAsync` is complete

Description

Class: `xPCParameter Class`

Event

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public event GetParamCompletedEventHandler GetParamCompleted` occurs when an asynchronous get parameter operation is complete.

xPCSignals.GetSignals

List of `xPCSignal` objects specified by array of signal identifiers

Syntax

```
public IList<xPCSignal> GetSignals(string[] arrayofBlockPath)
public IList<xPCSignal> GetSignals(int[] arrayOfSigId)
```

Description

Class: `xPCSignals Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public IList<xPCSignal> GetSignals(string[] arrayofBlockPath)` returns list of xPCSignal objects specified by array of signal identifiers. This method creates an ILIST of xPCSignal objects with an array of *`blockpath`*s. *`arrayofBlockPath`* is an array of strings that contains the full block path names to signals.

`public IList<xPCSignal> GetSignals(int[] arrayOfSigId)` returns the list of xPCSignal objects specified by an array of signal identifiers. This method creates an ILIST of xPCSignal objects with an array of signal identifiers. *`arrayOfSigId`* is an array of 32-bit integers that specifies an array of signal identifiers.

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCSignals.GetSignalsValue

Vector of signal values from array

Syntax

```
public double[] GetSignalsValue(int[] arrayOfSigId)
public double[] GetSignalsValue(IList<xPCSignals> arrayOfSigObjs)
```

Description

Class: `xPCSignals Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public double[] GetSignalsValue(int[] arrayOfSigId)` returns a vector of signal values from an array containing its signal identifiers. *`arrayOfSigId`* is an array of 32-bit signal identifiers. This method returns the vector as a double.

`public double[] GetSignalsValue(IList<xPCSignals> arrayOfSigObjs)` returns a vector of signal values from an IList that contains xPCSignals objects. This method returns the vector as a double.

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCSignal.GetValue

Value of signal at moment of request

Syntax

```
public virtual double GetValue()
```

Description

Class: `xPCSignal Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public virtual double GetValue()` returns signal value at moment of request.

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCTargetPC.Load

Load real-time application onto target computer

Syntax

```
public xPCApplication Load()
public xPCApplication Load(string DLMFileName)
```

Description

Class: `xPCTargetPC Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public xPCApplication Load()` loads a real-time application (`.dlm` file) onto the target computer. This method returns an xPCApplication object.

`public xPCApplication Load(string DLMFileName)` loads *`DLMFileName`* onto the target computer. *`DLMFileName`* is a string that specifies the full path name to the real-time application to load on the target computer. This method returns an xPCApplication object.

Exception

Exception	Condition
`ArgumentException`	*`DLMFileName`* is empty, contains only white spaces, or contains invalid characters.
`xPCException`	When problem occurs, query xPCException object `Reason` property.

Exception	Condition
`InvalidOperation-Exception`	*`DLMFileName`* is a NULL reference (empty in Visual Basic) or an empty string.
`NotSupportedException`	*`DLMFileName`* contains a colon (:) in the middle of the string.
`PathTooLongException`	The specified path, file name, or both in *`DLMFileName`* exceed the system-defined maximum length. For example, on Windows platforms, path names must be less than 248 characters. File names must be less than 260 characters.
`SecurityException`	Caller does not have required permission.
`UnauthorizedAccess-Exception`	System does not allow access to *`DLMFileName`*.

xPCTargetPC.LoadAsync

Asynchronous request to load real-time application onto target computer

Syntax

```
public void LoadAsync()
```

Description

Class: `xPCTargetPC Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void LoadAsync()` begins an asynchronous request to load a real-time application onto a target computer.

Exception

Exception	Condition
`InvalidOperation-Exception`	When another thread uses this method.

xPCTargetPC.LoadCompleted

Event when `xPCTargetPC.LoadAsync` is complete

Syntax

```
public event LoadCompletedEventHandler LoadCompleted
```

Description

Class: `xPCTargetPC Class`

Event

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public event LoadCompletedEventHandler LoadCompleted` occurs when an asynchronous load operation is complete.

xPCTargetPC.Loaded

Event after `xPCTargetPC.Load` is complete

Syntax

```
public event EventHandler Loaded
```

Description

Class: `xPCTargetPC Class`

Event

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public event EventHandler Loaded` occurs after real-time application onto the target computer is complete.

xPCTargetPC.Loading

Event before `xPCTargetPC.Load` starts

Syntax

```
public event EventHandler Loading
```

Description

Class: `xPCTargetPC Class`

Event

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public event EventHandler Loading` occurs before the loading of the real-time application starts on the target computer.

xPCParameters.LoadParameterSet

Load parameter values for real-time application

Syntax

```
public void LoadParameterSet(string fileName)
```

Description

Class: `xPCParameters Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void LoadParameterSet(string fileName)` loads parameter values for the real-time application in a file. *`fileName`* is a string that represents the file that contains the parameter values to be loaded.

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

CancelPropertyNotificationEventArgs Class

`CancelPropertyNotification` event data

Syntax

```
public class CancelPropertyNotificationEventArgs :
PropertyNotificationEventArgs
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class CancelPropertyNotificationEventArgs : PropertyNotificationEventArgs` contains data returned from the event of cancelling a property value change.

Properties

Properties	C# Declaration Syntax	Description
`Cancel`	`public bool Cancel {get; set;}`	Get or set value indicating whether or not to cancel event.
`NewValue`	`public Object NewValue {get;}`	Get new value of property.
`OldValue`	`public Object OldValue {get;}`	Get old value of property.
`PropertyName`	`public virtual string PropertyName {get;}`	Get name of property that changed.

ConnectCompletedEventArgs Class

`xPCTargetPC.ConnectCompleted` event data

Syntax

```
public class ConnectCompletedEventArgs : AsyncCompletedEventArgs
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class ConnectCompletedEventArgs : AsyncCompletedEventArgs` contains data returned from the event of asynchronously connecting to the target computer.

Properties

Properties	C# Declaration Syntax	Description
`Cancelled`	`public bool Cancelled {get;}`	Get value that indicates if an asynchronous operation has been cancelled.
`Error`	`public Exception Error {get;}`	Get value that indicates which error occurred during asynchronous operation.
`UserState`	`public Object UserState {get;}`	Get unique identifier for asynchronous task.

DisconnectCompletedEventArgs Class

`xPCTargetPC.DisconnectCompleted` event data

Syntax

```
public class DisconnectCompletedEventArgs : AsyncCompletedEventArgs
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class DisconnectCompletedEventArgs : AsyncCompletedEventArgs` contains data returned from the event of asynchronously disconnecting from the target computer.

Properties

Properties	C# Declaration Syntax	Description
`Cancelled`	`public bool Cancelled {get;}`	Get value that indicates if an asynchronous operation has been cancelled.
`Error`	`public Exception Error {get;}`	Get value that indicates which error occurred during asynchronous operation.
`UserState`	`public Object UserState {get;}`	Get unique identifier for asynchronous task.

GetDataCompletedEventArgs Class

`GetDataCompleted` event data

Syntax

```
public class GetDataCompletedEventArgs : AsyncCompletedEventArgs
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class GetDataCompletedEventArgs : AsyncCompletedEventArgs` contains data returned from the event of asynchronously completing a data access.

Properties

Properties	C# Declaration Syntax	Description
`Cancelled`	`public bool Cancelled {get;}`	Get value that indicates if an asynchronous operation has been cancelled.
`Error`	`public Exception Error {get;}`	Get value that indicates which error occurred during asynchronous operation.
`State`	`public Object State {get;}`	Optional. Get user-supplied state object.
`UserState`	`public Object UserState {get;}`	Get unique identifier for asynchronous task.

GetFileScSignalDataObjectCompletedEventArgs Class

`xPCDataFileScSignalObject.GetDataCompleted` event data

Syntax

```
public class GetFileScSignalDataObjectCompletedEventArgs :
GetDataCompletedEventArgs
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class GetFileScSignalDataObjectCompletedEventArgs : GetDataCompletedEventArgs` contains data returned from the event of completing an asynchronous data access to a file scope signal object.

Properties

Properties	C# Declaration Syntax	Description
`Cancelled`	`public bool Cancelled {get;}`	Get value that indicates if an asynchronous operation has been cancelled.
`Data`	`public double[] Data {get;}`	Get the signal data collected by file scope.
`Error`	`public Exception Error {get;}`	Get value that indicates which error occurred during asynchronous operation.
`FileScopeSignalObject`	`public bool IsScopeSignal {get;}`	Get reference to parent xPCFileScopeSignal object
`IsScopeSignal`	`public bool IsScopeSignal {get;}`	Get if signal is a scope signal (`true`) or a time signal (`false`).

Properties	C# Declaration Syntax	Description
`State`	`public Object State {get;}`	Optional. Get user-supplied state object.
`UserState`	`public Object UserState {get;}`	Get unique identifier for asynchronous task.

GetHostScSignalDataObjectCompletedEventArgs Class

`xPCDataHostScSignalObject.DataObjectCompleted` event data

Syntax

```
public class GetHostScSignalDataObjectCompletedEventArgs :
GetDataCompletedEventArgs
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class GetHostScSignalDataObjectCompletedEventArgs : GetDataCompletedEventArgs` contains data returned by the event of completing an asynchronous data access to a host scope signal object.

Properties

Properties	C# Declaration Syntax	Description
`Cancelled`	`public bool Cancelled {get;}`	Get value that indicates if an asynchronous operation has been cancelled.
`Data`	`public double[] Data {get;}`	Get the signal data collected by host scope
`Error`	`public Exception Error {get;}`	Get value that indicates which error occurred during asynchronous operation.
`IsScopeSignal`	`public bool IsScopeSignal {get;}`	Get if signal is a scope signal (`true`) or a time signal (`false`).

Properties	C# Declaration Syntax	Description
`ScopeSignalObject`	`public xPCScopeSignal ScopeSignalObject {get;}`	Get reference to parent xPCHostScopeSignal object
`State`	`public Object State {get;}`	Optional. Get user-supplied state object.
`UserState`	`public Object UserState {get;}`	Get unique identifier for asynchronous task.

GetLogDataCompletedEventArgs Class

`xPCDataLoggingObject.GetDataCompleted` event data

Syntax

```
public class GetLogDataCompletedEventArgs :
GetDataCompletedEventArgs
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class GetLogDataCompletedEventArgs : GetDataCompletedEventArgs` contains data returned by the event of completing an asynchronous data access to a data logging object.

Properties

Properties	C# Declaration Syntax	Description
`Cancelled`	`public bool Cancelled {get;}`	Get value that indicates if an asynchronous operation has been cancelled.
`Error`	`public Exception Error {get;}`	Get value that indicates which error occurred during asynchronous operation.
`Index`	`public int Index {get;}`	Get log index.
`LoggedData`	`public double[] LoggedData {get;}`	Get logged data.
`LogType`	`public xPClogType LogType {get;}`	Get log type as xPClogType.

Properties	C# Declaration Syntax	Description
`State`	`public Object State {get;}`	Optional. Get user-supplied state object.
`UserState`	`public Object UserState {get;}`	Get unique identifier for asynchronous task.

GetParamCompletedEventArgs Class

`xPCParameter.GetParamCompleted` event data

Syntax

```
public class GetParamCompletedEventArgs : AsyncCompletedEventArgs
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class GetParamCompletedEventArgs : AsyncCompletedEventArgs` contains data returned by the event of completing an asynchronous parameter access.

Properties

Properties	C# Declaration Syntax	Description
`Cancelled`	`public bool Cancelled {get;}`	Get value that indicates if an asynchronous operation has been cancelled.
`Error`	`public Exception Error {get;}`	Get value that indicates which error occurred during asynchronous operation.
`Result`	`public double[] Result {get;}`	Get data values of the xPCParameter object
`UserState`	`public Object UserState {get;}`	Get unique identifier for asynchronous task.

LoadCompletedEventArgs Class

`xPCTargetPC.LoadCompleted` event data

Syntax

```
public class LoadCompletedEventArgs : AsyncCompletedEventArgs
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class LoadCompletedEventArgs : AsyncCompletedEventArgs` contains data returned by the event of asynchronously loading a real-time application onto the target computer.

Properties

Properties	C# Declaration Syntax	Description
`Application`	`public xPCApplication Application {get;}`	Get reference to xPCApplication object.
`Cancelled`	`public bool Cancelled {get;}`	Get value that indicates if an asynchronous operation has been cancelled.
`Error`	`public Exception Error {get;}`	Get value that indicates which error occurred during asynchronous operation.
`UserState`	`public Object UserState {get;}`	Get unique identifier for asynchronous task.

PropertyNotificationEventArgs Class

`PropertyNotification` event data

Syntax

```
public class PropertyNotificationEventArgs :
PropertyChangedEventArgs
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class PropertyNotificationEventArgs : PropertyChangedEventArgs` contains data returned by the event of changing property values.

Properties

Properties	C# Declaration Syntax	Description
`NewValue`	`public Object NewValue {get;}`	Get new value of property.
`OldValue`	`public Object OldValue {get;}`	Get old value of property.
`PropertyName`	`public virtual string PropertyName {get;}`	Get name of property that changed.

RebootCompletedEventArgs Class

`xPCTargetPC.RebootCompleted` event data

Syntax

```
public class RebootCompletedEventArgs : AsyncCompletedEventArgs
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class RebootCompletedEventArgs : AsyncCompletedEventArgs` contains data returned by the event of asynchronously restarting the target computer.

Properties

Properties	C# Declaration Syntax	Description
`Cancelled`	`public bool Cancelled {get;}`	Get value that indicates if an asynchronous operation has been cancelled.
`Error`	`public Exception Error {get;}`	Get value that indicates which error occurred during asynchronous operation.
`UserState`	`public Object UserState {get;}`	Get unique identifier for asynchronous task.

SetParamCompletedEventArgs Class

`xPCParameter.SetParamCompleted` event data

Syntax

```
public class SetParamCompletedEventArgs : AsyncCompletedEventArgs
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class SetParamCompletedEventArgs : AsyncCompletedEventArgs` contains data returned by the event of asynchronously setting a parameter value.

Properties

Properties	C# Declaration Syntax	Description
`Cancelled`	`public bool Cancelled {get;}`	Get value that indicates if an asynchronous operation has been cancelled.
`Error`	`public Exception Error {get;}`	Get value that indicates which error occurred during asynchronous operation.
`NewValue`	`public Object NewValue {get;}`	Get new value of property.
`OldValue`	`public Object OldValue {get;}`	Get old value of property.
`UserState`	`public Object UserState {get;}`	Get unique identifier for asynchronous task.

UnloadCompletedEventArgs Class

`xPCTargetPC.UnloadCompleted` event data

Syntax

```
public class UnloadCompletedEventArgs : AsyncCompletedEventArgs
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class UnloadCompletedEventArgs : AsyncCompletedEventArgs` contains data returned by the event of asynchronously unloading the real-time application from the target computer.

Properties

Properties	C# Declaration Syntax	Description
`Cancelled`	`public bool Cancelled {get;}`	Get value that indicates if an asynchronous operation has been cancelled.
`Error`	`public Exception Error {get;}`	Get value that indicates which error occurred during asynchronous operation.
`UserState`	`public Object UserState {get;}`	Get unique identifier for asynchronous task.

xPCApplication Class

Access to real-time application loaded on target computer

Syntax

```
public sealed class xPCApplication : xPCBaseNotification
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public sealed class xPCApplication : xPCBaseNotification` initializes a new instance of the xPCApplication class.

Methods

Method	Description
xPCApplication.Start	Start real-time application execution
xPCApplication.Stop	Stop real-time application execution

Events

Events	Description
xPCApplication.Started	Event after `xPCApplication.Start` is complete
xPCApplication.Starting	Event before `xPCApplication.Start` executes
xPCApplication.Stopped	Event after `xPCApplication.Stop` is complete
xPCApplication.Stopping	Event before `xPCApplication.Stop` executes

Properties

Properties	C# Declaration Syntax	Description	Exception
`CPUOverload`	`public bool CPUOverload {get;}`	Get state of CPUOverload.	`xPCException` — When problem occurs, query xPCException object `Reason` property.
`ExecTime`	`public double ExecTime {get;}`	Get execution time.	`xPCException` — When problem occurs, query xPCException object `Reason` property.
`Logger`	`public xPCAppLogger Logger {get;}`	Get reference to the application logging object.	
`MaximumTeT`	`public double MaximumTeT {get;}`	Get the maximum time. The first element contains the maximum TET number; the second element contains how long it took to achieve the TET time.	`xPCException` — When problem occurs, query xPCException object `Reason` property.
`MinimumTeT`	`public double MinimumTeT {get;}`	Get the minimum time. The first element contains the minimum TET number; the second element contains how long it took to achieve the TET time.	`xPCException` — When problem occurs, query xPCException object `Reason` property.
`Name`	`public string Name {get;}`	Get the current name of the loaded real-time application	`xPCException` — When problem occurs, query xPCException object `Reason` property.
`Parameters`	`public xPCParameters Parameters {get;}`	Get reference to the xPCParameters object.	

Properties	C# Declaration Syntax	Description	Exception
`SampleTime`	`public double SampleTime {get; set;}`	Get or set Sample time	`xPCException` — When problem occurs, query xPCException object `Reason` property.
`Scopes`	`public xPCScopes Scopes {get;}`	Get collection of scopes assigned to the application	
`Signals`	`public xPCSignals Signals {get;}`	Get reference to xPCSignals object	
`Status`	`public xPCAppStatus Status {get;}`	Get simulation status. See xPCAppStatus Enumerated Data Type.	`xPCException` — When problem occurs, query xPCException object `Reason` property.
`StopTime`	`public double StopTime {get; set;}`	Get and set stop time	`xPCException` — When problem occurs, query xPCException object `Reason` property.
`Target`	`public xPCTargetPC Target {get;}`	Get reference to parent xPCTargetPC object.	

xPCAppLogger Class

Access to real-time application loggers

Syntax

```
public class xPCAppLogger : xPCApplicationObject
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class xPCAppLogger : xPCApplicationObject` initializes a new instance of the xPCAppLogger class.

Properties

Properties	C# Declaration Syntax	Description
`LogMode`	`public xPCLogMode LogMode {get; set;}`	Control which data points to log. See xPCLogMode Enumerated Data Type.
`LogModeValue`	`public int LogModeValue {get; set;}`	Get or set the value-equidistant logging. Set the value to the difference in signal values.
`MaxLogSamples`	`public int MaxLogSamples {get;}`	Get maximum number of samples that can be in log buffer.
`OutputLog`	`public xPCOutputLogger OutputLog {get;}`	Return a reference to the xPCOutputLogger object.
`StateLog`	`public xPCStateLogger StateLog {get;}`	Return a reference to the xPCStateLogger object.

Properties	C# Declaration Syntax	Description
`TETLog`	`public xPCTETLogger TETLog {get;}`	Return a reference to the xPCTETLogger object.
`TimeLog`	`public xPCTimeLogger TimeLog {get;}`	Return a reference to the xPCTimeLogger object.

xPCDataFileScSignalObject Class

Object that holds logged file scope signal data

Syntax

```
public class xPCDataFileScSignalObject : xPCFileScopeStream,
IxPCDataService
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class xPCDataFileScSignalObject : xPCFileScopeStream, IxPCDataService` accesses an object that holds logged file scope signal data.

Methods

Method	Description
xPCDataFileScSignalObject.G	Copy file scope signal data from target computer
xPCDataFileScSignalObject.G	Asynchronously copy file scope signal data from target computer

Events

Event	Description
xPCDataFileScSignalObject.G	Event when `xPCDataFileScSignalObject.GetDataAsync` is complete

Properties

Property	C# Declaration Syntax	Description
`ScopeSignal-Object`	`public xPCFileScopeSignal ScopeSignalObject {get;}`	Get parent scope signal xPCFileScopeSignal object.

xPCDataHostScSignalObject Class

Object that holds logged host scope signal data

Syntax

```
public class xPCDataHostScSignalObject :
xPCApplicationNotficationObject, IxPCDataService,
IxPCDataServiceAsync
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class xPCDataHostScSignalObject : xPCApplicationNotficationObject, IxPCDataService, IxPCDataServiceAsync` accesses an object that holds logged host scope signal data.

Methods

Method	Description
xPCDataHostScSignalObject.	Copy host scope signal data from target computer
xPCDataHostScSignalObject.	Asynchronously copy host scope signal data from target computer

Events

Event	Description
xPCDataHostScSignalObject.	Event when `xPCDataHostScSignalObject.GetDataAsync` is complete

Properties

Property	C# Declaration Syntax	Description
`Decimation`	`public int Decimation {get; set;}`	A number *n*, where every *n*th sample is acquired in a scope window.
`NumSamples`	`public int NumSamples {get; set;}`	Get or set 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. It then has zeroes for the remaining uncollected data. Note what type of data you are collecting, it is possible that your data contains zeroes. For file scopes, this parameter works with the autorestart setting. If autorestart is enabled, the file scope collects data up to `NumSamples`, then starts over again, overwriting the buffer. If autorestart is disabled, the file scope collects data only up to `NumSamples`, then stops.
`ScopeSignal-Object`	`public xPCHostScopeSignal ScopeSignalObject {get;}`	Get parent scope signal xPCHostScopeSignal object.
`Startindex`	`public int StartIndex {get; set;}`	Get and set the index of the first sample to retrieve from the log.

xPCDataLoggingObject Class

Object that holds logged data

Syntax

```
public class xPCDataLoggingObject : xPCApplicationNotficationObject,
IxPCDataService, xPCDataServiceAsync
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class xPCDataLoggingObject : xPCApplicationNotficationObject, IxPCDataService, xPCDataServiceAsync` accesses an object that holds logged data.

Methods

Method	Description
xPCDataLoggingObject.GetD	Copy signal data from target computer
xPCDataLoggingObject.GetD	Asynchronously copy signal data from target computer

Events

Event	Description
xPCDataLoggingObject.GetD	Event when `xPCDataLoggingObject.GetDataAsync` is complete

Properties

Property	C# Declaration Syntax	Description
`Decimation`	`public int Decimation {get; set;}`	A number *n*, where every *n*th sample is acquired in a scope window.

Property	C# Declaration Syntax	Description
`LogId`	`public int LogId {get;}`	
`NumSamples`	`public int NumSamples {get; set;}`	Get or set 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. It then has zeroes for the remaining uncollected data. Note what type of data you are collecting, it is possible that your data contains zeroes. For file scopes, this parameter works with the autorestart setting. If autorestart is enabled, the file scope collects data up to `NumSamples`, then starts over again, overwriting the buffer. If autorestart is disabled, the file scope collects data only up to `NumSamples`, then stops.
`Startindex`	`public int StartIndex {get; set;}`	Get and set the index of the first sample to retrieve from the log.

xPCDirectoryInfo Class

Access folders and subfolders of target computer file system

Syntax

```
public class xPCDirectoryInfo : xPCFileSystemInfo
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class xPCDirectoryInfo : xPCFileSystemInfo` accesses folders and subfolders of target computer file system.

Constructor

Constructor	Description
xPCDirectoryInfo	Construct new instance of the xPCDirectoryInfo class on specified path

Methods

Method	Description
xPCDirectoryInfo.Create	Create folder
xPCDirectoryInfo.Delete	Delete empty xPCDirectoryInfo object
xPCDirectoryInfo.GetDirector	Subfolders of current folder
xPCDirectoryInfo.GetFiles	File list from current folder
xPCDirectoryInfo.GetFileSyst	File system information for files and subfolders in folder

Properties

Property	C# Declaration Syntax	Description	Exception
CreationTime	public override DateTime CreationTime {get;}	Get creation time of the current FileSystemInfo object.	xPCException — When problem occurs, query xPCException object Reason property.
Exists	public override bool Exists {get;}	Get a Boolean value to indicate existence of folder. A value of 1 indicates existent, 0 indicates nonexistent.	xPCException — When problem occurs, query xPCException object Reason property.
Extension	public string Extension {get;}	Get string that represents the extension part of the file.	
FullName	public virtual string FullName {get;}	Get full path name of the folder or file.	
Name	public override string Name {get;}	Get the name of this xPCDirectoryInfo instance as a string.	xPCException — When problem occurs, query xPCException object Reason property.
Parent	public xPCDirectoryInfo Parent {get;}	Get the parent folder of a specified subfolder.	xPCException — When problem occurs, query xPCException object Reason property.
Root	public xPCDirectoryInfo Root {get;}	Get the root portion of a path.	xPCException — When problem occurs, query xPCException object Reason property.

xPCDriveInfo Class

Information for target computer drive

Syntax

```
public class xPCDriveInfo
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class xPCDriveInfo` accesses information on a target computer drive.

Constructor

Constructor	Description
xPCDriveInfo	Initialize new instance of xPCDriveInfo class

Methods

Method	Description
xPCDriveInfo.Refresh	Synchronize with file drives on target computer

Properties

Property	C# Declaration Syntax	Description	Exception
`Available-Freespace`	`public long AvailableFreeSpace {get;}`	Indicate amount of available free space on drive.	`xPCException` — When problem occurs, query xPCException object `Reason` property.

Property	C# Declaration Syntax	Description	Exception
DriveFormat	public string DriveFormat {get;}	Get name of file system type, such as FAT16 or FAT32.	xPCException — When problem occurs, query xPCException object Reason property.
Name	public string Name {get;}	Get name of drive.	xPCException — When problem occurs, query xPCException object Reason property.
Root-Directory	public xPCDirectoryInfo RootDirectory {get;}	Get root folder of drive.	xPCException — When problem occurs, query xPCException object Reason property.
TotalSize	public long TotalSize {get;}	Get total size of drive in bytes.	xPCException — When problem occurs, query xPCException object Reason property.
VolumeLabel	public string VolumeLabel {get;}	Get volume label of drive.	xPCException — When problem occurs, query xPCException object Reason property.

xPCException Class

Information for `xPCException`

Syntax

```
public class xPCException : Exception, ISerializable
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class xPCException : Exception, ISerializable` accesses information on Simulink Real-Time exceptions.

Constructor

Constructor	Description
xPCException	Construct new instance of xPCException class

Properties

Property	C# Declaration Syntax	Description
`Data`	`public virtual IDictionary Data {get;}`	Get collection of key/value pairs that provide additional user-defined information about the exception.
`HelpLink`	`public virtual string HelpLink {get; set;}`	Get or set link to the help file associated with this exception.
`InnerException`	`public Exception InnerException {get;}`	Get Exception instance that caused the current exception.
`Message`	`public override string Message {get;}`	Get exception message. Overrides `Exception.Message` property.

Property	C# Declaration Syntax	Description
Reason	public xPCExceptionReason Reason {get;}	Get xPCExceptionReason reason. See xPCExceptionReason Enumerated Data Type.
Source	public virtual string Source {get; set;}	Get or set name of real-time application or object that causes the error.
StackTrace	public virtual string StackTrace {get;}	Get string representation of the frames on the call stack at the time the method emits the current exception.
TargetPCObject	public xPCTargetPC TargetPCObject {get;}	Get xPCTargetPC object that raised the error.
TargetSite	public MethodBase TargetSite {get;}	Get method that emits the current exception.

xPCFileInfo Class

Access to file and `xPCFileStream` objects

Syntax

```
public class xPCDriveInfo
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class xPCDriveInfo` accesses information on a target computer drive.

Constructor

Constructor	Description
xPCFileInfo	Construct new instance of xPCFileInfo class

Methods

Method	Description
xPCFileInfo.CopyToHost	Copy file from target computer file system to development computer file system
xPCFileInfo.Create	Create file in specified path name
xPCFileInfo.Delete	Permanently delete file on target computer
xPCFileInfo.Open	Open file
xPCFileInfo.OpenRead	Create read-only xPCFileStream object
xPCFileInfo.Rename	Rename file
xPCFileInfo	Construct new instance of xPCFileInfo class

Properties

Property	C# Declaration Syntax	Description
`Directory`	`public xPCDirectoryInfo Directory {get;}`	Get an xPCDirectoryInfo object.
`DirectoryName`	`public string DirectoryName {get;}`	Get a string that represents the full folder path name.
`Exists`	`public override bool Exists {get;}`	Get value that indicates whether a file exists.
`Length`	`public long Length {get;}`	Get the size, in bytes, of the current file.
`Name`	`public override string Name {get;}`	Get the name of the file.

xPCFileScope Class

Access to file scopes

Syntax

```
public class xPCFileScope : xPCScope
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class xPCFileScope : xPCScope` initializes a new instance of the xPCFileScope class.

Methods

The xPCFileScope class inherits methods from `xPCScope Class`.

Events

The xPCFileScope class inherits events from `xPCScope Class`.

Properties

The xPCFileScope class inherits its other properties from xPCScope Class.

Property	C# Declaration Syntax	Description	Exception
AutoRestart	public bool AutoRestart {get; set;}	Get or set the file scope autorestart setting. AutoRestart is a Boolean. Values are 'on' and 'off'.	xPCException — When problem occurs, query xPCException object Reason property.
DataTime-Object	public xPCDataHostScSignalObje DataTimeObject {get;}	Get data time object.	xPCException — When problem occurs, query xPCException object Reason property.
DynamicMode	public bool DynamicMode {get; set;}	Get or set ability to dynamically create multiple log files for file scopes. Values are 'on' and 'off' . By default, the value is 'off'.	xPCException — When problem occurs, query xPCException object Reason property.
FileMode	public SCFILEMODE FileMode {get; set;}	Get or set write mode of file. See xPCFileMode Enumerated Data Type.	xPCException — When problem occurs, query xPCException object Reason property.
FileName	public string FileName {get; set;}	Get or set file name for scope.	
MaxWrite-FileSize	public uint MaxWriteFileSize {get; set;}	Get or set the maximum file size in bytes allowed before incrementing to the next file. When the size of a log file reaches MaxWriteFileSize, the software creates a subsequently numbered file name, and continues logging data to that file,	xPCException — When problem occurs, query xPCException object Reason property.

Property	C# Declaration Syntax	Description	Exception
		up until the highest log file number you have specified. If the software cannot create additional log files, it overwrites the first log file. This value must be a multiple of `WriteSize`. Default is 536870912.	
`Signals`	`public xPCTarget-ScopeSignalCollection Signals {get;}`	Get collection of file scope signals (xPCFileScope-SignalCollection) assigned to this scope object.	
`Trigger-Signal`	`public xPCTgtScopeSignal TriggerSignal {get; set;}`	Get or set file scope signal (xPCFileScopeSignal) used to trigger the scope.	`xPCException` — When problem occurs, query xPCException object `Reason` property.
`WriteSize`	`public int WriteSize {get; set;}`	Get or set the unit number of bytes for memory buffer writes. The memory buffer accumulates data in multiples of write size. *`WriteSize`* must be multiple of 512.	`xPCException` — When problem occurs, query xPCException object `Reason` property.

xPCFileScopeCollection Class

Collection of `xPCFileScope` objects

Syntax

```
public class xPCFileScopeCollection :
xPCScopeCollection<xPCFileScope>
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class xPCFileScopeCollection : xPCScopeCollection<xPCFileScope>` initializes collection of xPCFileScope objects.

Methods

Method	Description
xPCFileScopeCollection.Add	Create xPCFileScope object with the next available scope ID as key
xPCFileScopeCollection.Refre	Synchronize with file scopes on target computer
xPCFileScopeCollection.Start	Start all file scopes in one call
xPCFileScopeCollection.Stop	Stop all file scopes in one call

xPCFileScopeSignal Class

Access to file scope signals

Syntax

```
public class xPCFileScopeSignal : xPCScopeSignal
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class xPCFileScopeSignal : xPCScopeSignal` initializes access to file scope signals.

Properties

Property	C# Declaration Syntax	Description
`FileScopeSignal-DataObject`	`public xPCDataFileScSignalObject FileScopeSignalDataObject {get;}`	Get the data xPCDataFileScSignalObject object associated with this xPCFileScopeSignal object.
`Scope`	`public xPCFileScope Scope {get;}`	Get parent file scope xPCFileScope object.

xPCFileScopeSignalCollection Class

Collection of `xPCFileScopeSignal` objects

Syntax

```
public class xPCFileScopeSignalCollection :
xPCScopeSignalCollection<xPCFileScopeSignal>
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class xPCFileScopeSignalCollection : xPCScopeSignalCollection<xPCFileScopeSignal>` initializes collection of xPCFileScopeSignal objects.

Methods

Method	Description
xPCFileScopeSignalCollectior	Add signals to file scope
xPCFileScopeSignalCollectior	Synchronize with signals for associated scope on target computer

Properties

Property	C# Declaration Syntax	Description	Exception
`Item`	`public xPCFileScopeSignal Item[string blkpath] {get;}`	Get xPCFileScopeSignal object from signal name (*`blkpath`*). *`blkpath`* is the signal name that represents a signal object added to its	`xPCException` — When problem occurs, query xPCException object `Reason` property.

Property	C# Declaration Syntax	Description	Exception
		parent xPCHostScope object. This property returns the file scope signal object as type xPCFileScopeSignal.	

xPCFileStream Class

Access `xPCFileStream` objects

Syntax

```
public class xPCFileStream : IDisposable
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class xPCFileStream : IDisposable` initializes xPCFileStream objects. These objects expose the file stream around a file.

Constructor

Constructor	Description
xPCFileStream	Construct new instance of xPCFileStream class

Methods

Method	Constructor
xPCFileStream.Close	Close current stream
xPCFileStream.Read	Read block of bytes from stream and write data to buffer
xPCFileStream.Write	Write block of bytes to file stream
xPCFileStream.WriteByte	Write byte to current position in file stream

Property

Property	C# Declaration Syntax	Description	Exception
`Length`	`public long Length {get;}`	Get length of file stream.	`xPCException` — When problem occurs, query

Property	C# Declaration Syntax	Description	Exception
			xPCException object `Reason` property.

xPCFileSystem Class

File system drives and folders

Syntax

```
public class xPCFileSystem
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class xPCFileSystem` initializes file system drive and folder objects.

Methods

Method	Description
xPCFileSystem.Create	Create folder
xPCFileSystem.GetCurrentD	Current working folder for real-time application
xPCFileSystem.GetDrives	Drive names for the logical drives on the target computer
xPCFileSystem.RemoveFile	Remove file name from target computer
xPCFileSystem.SetCurrentDi	Current folder

xPCFileSystemInfo Class

File system information

Syntax

```
public abstract class xPCFileSystemInfo
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public abstract class xPCFileSystemInfo` initializes file system information objects.

Constructor

Constructor	Description
xPCFileSystemInfo	Initialize new instance of xPCFileSystemInfo class

Methods

Method	Description
xPCFileSystemInfo.Delete	Delete current folder

Properties

Property	C# Declaration Syntax	Description
`CreationTime`	`public DateTime CreationTime {get;}`	Get creation time of current FileSystemInfo object.
`Exists`	`public abstract bool Exists {get;}`	Get value that indicates existence of file or folder.

Property	C# Declaration Syntax	Description
`Extension`	`public string Extension {get;}`	Get string that represents file extension.
`FullName`	`public virtual string FullName {get;}`	Get full path name of file or folder.
`Name`	`public abstract string Name {get;}`	Get name of folder.

xPCHostScope Class

Access to host scopes

Syntax

```
public class xPCHostScope : xPCScope
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class xPCHostScope : xPCScope` initializes a new instance of the xPCHostScope class.

Methods

The xPCHostScope class inherits methods from `xPCScope Class`.

Events

The xPCHostScope class inherits events from `xPCScope Class`.

Properties

The xPCHostScope class inherits its other properties from `xPCScope Class`.

Property	C# Declaration Syntax	Description	Exception
`DataTimeObject`	`public xPCDataHostScSignalObject DataTimeObject {get;}`	Get host scope time data object xPCDataHostScSignalObject associated with this scope.	

Property	C# Declaration Syntax	Description	Exception
`Signals`	`public xPCTarget-ScopeSignal-Collection Signals {get;}`	Get collection of host scope signals (xPCHost-ScopeSignalCollection) assigned to this scope object.	
`Trigger-Signal`	`public xPCTgtScope-Signal TriggerSignal {get; set;}`	Get or set host scope signal (xPCHostScope-Signal) used to trigger the scope.	`xPCException` — When problem occurs, query xPCException object `Reason` property.

xPCHostScopeCollection Class

Collection of `xPCHostScope` objects

Syntax

```
public class xPCHostScopeCollection :
xPCScopeCollection<xPCHostScope>
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class xPCHostScopeCollection : xPCScopeCollection<xPCHostScope>` initializes collection of xPCHostScope objects.

Methods

Method	Description
xPCHostScopeCollection.Add	Create xPCHostScope object with the next available scope ID as key
xPCHostScopeCollection.Refr	Refresh host scope object state
xPCHostScopeCollection.Star	Start all host scopes in one call
xPCHostScopeCollection.Stop	Stop all host scopes in one call

xPCHostScopeSignal Class

Access to host scope signals

Syntax

```
public class xPCHostScopeSignal : xPCScopeSignal
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class xPCHostScopeSignal : xPCScopeSignal` initializes access to host scope signals.

Properties

Property	C# Declaration Syntax	Description
`HostScopeSignal-DataObject`	`public xPCDataHostScSignalObject HostScopeSignalDataObject {get;}`	Get host scope signal data object.
`Scope`	`public xPCHostScope Scope {get;}`	Get host scope.

xPCHostScopeSignalCollection Class

Collection of `xPCHostScopeSignal` objects

Syntax

```
public class xPCHostScopeSignal : xPCScopeSignal
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class xPCHostScopeSignal : xPCScopeSignal` represents a collection of xPCHostScopeSignal objects.

Methods

Method	Description
xPCHostScopeSignalCollectio	Create xPCHostScopeSignal object
xPCHostScopeSignalCollectio	Synchronize signals for associated host scopes on target computer

Properties

Property	C# Declaration Syntax	Description	Exception
`Item`	`public xPCHostScopeSignal Item[string blkpath] {get;}`	Get xPCHostScopeSignal object from signal name (*`blkpath`*). *`blkpath`* is the signal name that represents a signal object added to its	`xPCException` — When problem occurs, query xPCException object `Reason` property.

Property	C# Declaration Syntax	Description	Exception
		parent xPCHostScope object. This property returns the file scope signal object as type xPCHostScopeSignal.	

xPCLog Class

Base data logging class

Syntax

```
public abstract class xPCLog : xPCApplicationObject
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public abstract class xPCLog : xPCApplicationObject` represents the base data logging class.

Properties

Properties	C# Declaration Syntax	Description
`IsEnabled`	`public abstract bool IsEnabled {get;}`	Get whether to enable or disable logging.
`NumLogSamples`	`public int NumLogSamples {get;}`	Get number of samples in log buffer.
`NumLogWraps`	`public int NumLogWraps {get;}`	Get number of times log buffer wraps.

xPCOutputLogger Class

Access to output logger

Syntax

```
public class xPCOutputLogger : xPCLog
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class xPCOutputLogger : xPCLog` initializes a new instance of the xPCOutputLogger class.

Properties

The xPCOutputLogger class inherits its other properties from `xPCLog Class`.

Properties	C# Declaration Syntax	Description
`DataLoggingObjects`	`public IList<xPCDataLoggingObject DataLoggingObjects {get;}`	Get ILIST of application data logging objects.
`IsEnabled`	`public override bool IsEnabled {get;}`	Get whether to enable or disable logging. Overrides xPCLog.IsEnabled.
`Item`	`public xPCDataLoggingObject Item[int index ] {get;}`	Get xPCDataLogging object specified by index (*`index`*). *`index`* is the index to the specified logging output. This property returns an object of type xPCDataLoggingObject.

Properties	C# Declaration Syntax	Description
`NumOutputs`	`public int NumOutputs {get;}`	Return a reference to the xPCOutputLogger object.

xPCParameter Class

Single run-time tunable parameter

Syntax

```
public class xPCParameter : xPCApplicationNotficationObject
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class xPCParameter : xPCApplicationNotficationObject` initializes a new instance of the xPCParameter class. An xPCParameter object represents a single specific real-time application parameter. You can tune the parameter using xPCParameter objects.

Methods

Method	Description
xPCParameter.GetParam	Get parameter values from target computer
xPCParameter.GetParamAsy	Asynchronous request to get parameter values from target computer
xPCParameter.SetParam	Change value of parameter on target computer
xPCParameter.SetParamAsy	Asynchronous request to change parameter value on target computer

Events

Event	Description
xPCParameter.GetParamCon	Event when `xPCParameter.GetParamAsync` is complete
xPCParameter.SetParamCon	Event when `xPCParameter.SetParamAsync` is complete

Properties

Property	C# Declaration Syntax	Description	Exception
`BlockPath`	`public string BlockPath {get;}`	Get the full block path name of the parameter for an instance of an xPCParameter object.	
`DataType`	`public string DataType {get;}`	Get the Simulink type, as a string, of the parameter for an instance of an xPCParameter object.	
`Dimensions`	`public int[] Dimensions {get;}`	Get an array that contains elements of dimension lengths.	
`Name`	`public string Name {get;}`	Get the name of the parameter to an instance of an xPCParameter	
`ParameterId`	`public int ParameterId {get;}`	Get the numerical index (identifier) that maps to an instance of an xPCParameter object.	
`Rank`	`public int Rank {get;}`	Get the number of dimensions of the parameter	
`Value`	`public Array Value {get; set;}`	Get and set the parameter value.	`xPCException` — When problem occurs, query xPCException object `Reason` property.

xPCParameters Class

Access run-time parameters

Syntax

```
public class xPCParameters : xPCApplicationObject
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class xPCParameters : xPCApplicationObject` initializes a new instance of the xPCParameters class. An xPCParameters object is a container to access run time parameters.

Methods

Method	Description
xPCParameters.LoadParame	Load parameter values for real-time application
xPCParameters.Refresh	Refresh state of object
xPCParameters.SaveParamet	Save parameter values of real-time application

Properties

Property	C# Declaration Syntax	Description
`NumParameters`	`public int NumParameters {get;}`	Get the total number of tunable parameters in the real-time application.

Property	C# Declaration Syntax	Description
`Item`	`public xPCParameter Item[int paramIdx] {get;}` or `public xPCParameter Item[string blkName, string paramName] {get;}`	Return reference to xPCParameter object specified by its parameter identifier (*paramIdx*) or parameter name (*paramname*). *paramIdx* is a 32-bit integer parameter identifier that represents the actual signal. *blkName* is a string that specifies the block path name for the actual block that contains the parameter. *paramName* is a string that specifies the parameter name. This method returns the xPCParameter object that represents the actual parameter.

xPCScope Class

Access Simulink Real-Time scopes

Syntax

```
public abstract class xPCScope : xPCApplicationNotficationObject
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public abstract class xPCScope : xPCApplicationNotficationObject` initializes a new instance of the xPCScope class.

Methods

Method	Description
xPCScope.Start	Start scope
xPCScope.Stop	Stop scope
xPCScope.Trigger	Software-trigger start of data acquisition for scopes

Events

Event	Description
xPCScope.ScopeStarted	Event after `xPCScope.Start` is complete
xPCScope.ScopeStarting	Event before `xPCScope.Start` executes
xPCScope.ScopeStopped	Event after `xPCScope.Stop` is complete
xPCScope.ScopeStopping	Event before `xPCScope.Stop` executes

Properties

Property	C# Declaration Syntax	Description	Exception
`Decimation`	`public int Decimation {get; set;}`	Get or set a number *n*, where every *n*th sample is acquired in a scope window.	`xPCException` — When problem occurs, query xPCException object `Reason` property.
`NumPrePost-Samples`	`public int NumPrePostSamples {get; set;}`	Get or set 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'`, changing this property does not change data acquisition.	`xPCException` — When problem occurs, query xPCException object `Reason` property.
`NumSamples`	`public int NumSamples {get; set;}`	Get or set 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. It then has zeroes for the remaining uncollected data. Note what type of data you are collecting, it is possible that your data contains zeroes.	`xPCException` — When problem occurs, query xPCException object `Reason` property.

Property	C# Declaration Syntax	Description	Exception
		For file scopes, this parameter works with the autorestart setting. If autorestart is enabled, the file scope collects data up to `NumSamples`, then starts over again, overwriting the buffer. If autorestart is disabled, the file scope collects data only up to `NumSamples`, then stops.	
`ScopeId`	`public int ScopeId {get;}`	A numeric index, unique for each scope.	
`Status`	`public SCSTATUS Status {get;}`	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'`.	`xPCException` — When problem occurs, query xPCException object `Reason` property.
`TriggerAnySign`	`public int TriggerAnySignal {get; set;}`	Get or set `xPCSignal Class` object for trigger signal. If `TriggerMode` is `'Signal'`, this signal triggers the scope even if it was not added to the scope.	`xPCException` — When problem occurs, query xPCException object `Reason` property.

Property	C# Declaration Syntax	Description	Exception
TriggerLevel	public double TriggerLevel {get; set;}	Get or set trigger level. If TriggerMode is 'Signal', indicates the value the signal has to cross to trigger the scope and start acquiring data. You can cross the trigger level with either a rising or falling signal.	xPCException — When problem occurs, query xPCException object Reason property.
TriggerMode	public SCTRIGGERMODE TriggerMode {get; set;}	Get or set trigger mode for a scope. Valid values are 'FreeRun' (default), 'Software', 'Signal', and 'Scope'.	xPCException — When problem occurs, query xPCException object Reason property.
TriggerScope	public int TriggerScope {get; set;}	If TriggerMode is 'Scope', identifies the scope to use for a trigger. You can set a scope to trigger when another scope is triggered. You do this operation by setting the slave scope property TriggerScope to the scope index of the master scope.	xPCException — When problem occurs, query xPCException object Reason property.
TriggerScope-Sample	public int TriggerScopeSample {get; set;}	If TriggerMode is 'Scope', specifies the number of samples the triggering scope is to acquire before triggering a second scope. This value must be nonnegative.	xPCException — When problem occurs, query xPCException object Reason property.

Property	C# Declaration Syntax	Description	Exception
TriggerSlope	public TRIGGERSLOPE {get; set;}	If TriggerMode is 'Signal', indicates whether the trigger is on a rising or falling signal. Values are of type SLTRIGGERSLOPE: SLTRIGGERSLOPE.EITHEI (default), SLTRIGGERSLOPE.RISIN(and SLTRIGGERSLOPE.FALLII This property returns the value SCTRIGGERSLOPE.	xPCException — When problem occurs, query xPCException object Reason property.
Type	public string Type {get;}	Get scope type as a string.	

xPCScopeCollectionEventArgs Class

`xPCScopeCollection.Added` event data

Syntax

```
public class xPCScopeCollectionEventArgs : EventArgs
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class xPCScopeCollectionEventArgs : EventArgs` contains data returned by the event of adding a scope to a scope collection.

Properties

Properties	C# Declaration Syntax	Description
`Scope`	`public xPCScope Scope {get;}`	Get xPCScope object you added.

xPCScopeRemCollectionEventArgs Class

`xPCScopeCollection.Removed` event data

Syntax

```
public class xPCScopeRemCollectionEventArgs : EventArgs
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class xPCScopeRemCollectionEventArgs : EventArgs` contains data returned by the event of removing a scope from a scope collection.

Properties

Properties	C# Declaration Syntax	Description
`ScopeNumber`	`public int ScopeNumber {get;}`	Get scope number of the scope that you have removed.

xPCScopeSignalCollectionEventArgs Class

`xPCScopeSignalCollection.Added` event data

Syntax

```
public class xPCScopeSignalCollectionEventArgs : EventArgs
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class xPCScopeSignalCollectionEventArgs : EventArgs` contains data returned by the event of adding a signal to a scope signal collection.

Properties

Properties	C# Declaration Syntax	Description
`Scope`	`public xPCScope Scope {get;}`	Get parent xPCScope object
`Signal`	`public xPCSignal Signal {get;}`	Get xPCSignal object that you added to collection.

xPCScopes Class

Access scope objects

Syntax

```
public class xPCScopes : xPCApplicationObject
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class xPCScopes : xPCApplicationObject` initializes a new instance of the xPCScopes class.

Methods

Method	Description
xPCScopes.RefreshAll	Synchronize with all scopes on target computer

Properties

Property	C# Declaration Syntax	Description
`FileScopes`	`public xPCFileScopeCollection FileScopes {get;}`	Get collection of file scopes (xPCFileScopeCollection).
`HostScopes`	`public xPCHostScopeCollection HostScopes {get;}`	Get collection of host scopes (xPCHostScopeCollection).

Property	C# Declaration Syntax	Description
`ScopeObjectDict`	`public IDictionary<int, xPCScope> ScopeObjectDict {get;}`	Get entire scopes object as a Dictionary object.
`ScopeObjectList`	`public IList<xPCScope> ScopeObjectList {get;}`	Get entire scopes object as a list.
`TargetScopes`	`public xPCTargetScopeCollection TargetScopes {get;}`	Get collection of target scopes (xPCTargetScopeCollection).

xPCSignal Class

Access signal objects

Syntax

```
public class xPCSignal : xPCApplicationObject
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class xPCSignal : xPCApplicationObject` initializes a new instance of the xPCSignal class.

Methods

Method	Description
xPCSignal.GetValue	Value of signal at moment of request
xPCSignal.TryGetValue	Status of get signal value at moment of request

Properties

Property	C# Declaration Syntax	Description
`BlockPath`	`public virtual string BlockPath {get;}`	Get block path name (signal name) of the signal.
`DataType`	`public virtual string DataType {get;}`	Get Simulink data type name.
`Label`	`public virtual string Label {get;}`	Get label of signal. If no label is associated with the signal, this property returns an empty string.

Property	C# Declaration Syntax	Description
`SignalId`	`public virtual int SignalId {get;}`	Get numeric identifier that represents the signal object.
`UserData`	`public Object UserData {get; set;}`	Get and set user-defined object that you can use to store and retrieve additional information.
`Width`	`public virtual int Width {get;}`	Get signal width.

xPCSignals Class

Access signal objects

Syntax

```
public class xPCSignals : xPCApplicationObject
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class xPCSignals : xPCApplicationObject` initializes a new instance of the xPCSignals class.

Methods

Method	Description
xPCSignals.GetSignals	List of xPCSignal objects specified by array of signal identifiers
xPCSignals.GetSignalsValue	Vector of signal values from array
xPCSignals.Refresh	Refresh state of object

Properties

Property	C# Declaration Syntax	Description	Exception
`NumSignal`	`public int NumSignals {get;}`	Get total numbers of signals available in real-time application.	

Property	C# Declaration Syntax	Description	Exception
`this`	`public xPCSignal Item[int signalIdx ] {get;}` or `public xPCSignal Item[string blkPath ] {get;}`	Return reference to xPCSignal object specified by its signal identifier (*`signalIdx`*) or signal name (*`blkPath`*). *`signalIdx`* is a 32–bit integer that identifies the signal. *`blkPath`* is a string that specifies the block path name for the signal.	`xPCException` — When problem occurs, query xPCException object `Reason` property. `ArgumentNullException` — *`signalIdx`* or *`blkPath`* is NULL reference.

xPCStateLogger Class

Access to state log

Syntax

```
public class xPCStateLogger : xPCLog
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class xPCStateLogger : xPCLog` initializes a new instance of the xPCStateLogger class.

Properties

The xPCStateLogger class inherits its other properties from `xPCLog Class`.

Property	C# Declaration Syntax	Description
`DataLogging-Objects`	`public IList<xPCDataLoggingObject> DataLoggingObjects {get;}`	Get collection of xPCDataLoggingObject items available for state logging.
`IsEnabled`	`public override bool IsEnabled {get;}`	Get whether to enable or disable logging. Overrides xPCLog.IsEnabled.
`Item`	`public xPCDataLoggingObject Item[ int index ] {get;}`	Get reference to the xPCLoggingObject that corresponds to *`index`* (state index). *`index`* is a 32–bit integer.
`NumStates`	`public int NumStates {get;}`	Get the number of states.

xPCTargetPC Class

Access target computer

Syntax

```
public xPCTargetPC()
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

public xPCTargetPC() initializes a new instance of the xPCTargetPC class.

Constructor

Constructor	Description
xPCTargetPC	Construct xPCTargetPC object.

Methods

Method	Description
xPCTargetPC.Connect	Establish connection to target computer
xPCTargetPC.ConnectAsync	Asynchronous request for target computer connection
xPCTargetPC.Disconnect	Disconnect from target computer
xPCTargetPC.DisconnectAsy	Asynchronous request to disconnect from target computer
xPCTargetPC.Dispose	Clean up used resources
xPCTargetPC.Load	Load real-time application onto target computer

Method	Description
xPCTargetPC.LoadAsync	Asynchronous request to load real-time application onto target computer
xPCTargetPC.Ping	Test communication between development and target computers
xPCTargetPC.Reboot	Restart target computer
xPCTargetPC.RebootAsync	Asynchronous request to restart target computer
xPCTargetPC.tcpPing	Determine TCP/IP accessibility of remote computer
xPCTargetPC.Unload	Unload real-time application from target computer
xPCTargetPC.UnloadAsync	Asynchronous request to unload real-time application from target computer

Events

Event	Description
xPCTargetPC.ConnectComple	Event when `xPCTargetPC.ConnectAsync` is complete
xPCTargetPC.Connected	Event after `xPCTargetPC.Connect` is complete
xPCTargetPC.Connecting	Event before `xPCTargetPC.Connect` starts
xPCTargetPC.DisconnectCom	Event when `xPCTargetPC.DisconnectAsync` is complete
xPCTargetPC.Disconnected	Event after `xPCTargetPC.Disconnect` is complete
xPCTargetPC.Disconnecting	Event before `xPCTargetPC.Disconnect` starts
xPCTargetPC.Disposed	Event after `xPCTargetPC.Dispose` is complete
xPCTargetPC.LoadCompleted	Event when `xPCTargetPC.LoadAsync` is complete
xPCTargetPC.Loaded	Event after `xPCTargetPC.Load` is complete
xPCTargetPC.Loading	Event before `xPCTargetPC.Load` starts
xPCTargetPC.RebootComplet	Event when `xPCTargetPC.RebootAsync` is complete
xPCTargetPC.Rebooted	Event after `xPCTargetPC.Reboot` is complete
xPCTargetPC.Rebooting	Event before `xPCTargetPC.Reboot` starts
xPCTargetPC.UnloadComple	Event when `xPCTargetPC.UnloadAsync` is complete
xPCTargetPC.Unloaded	Event after `xPCTargetPC.Unload` is complete
xPCTargetPC.Unloading	Event before `xPCTargetPC.Unload` starts

Properties

Property	C# Declaration Syntax	Description	Exception
`Application`	`public xPCApplication Application {get;}`	Get reference to an `xPCApplication` object that you can use to interface with the real-time application. If no communication is established, the property returns a NULL object.	
`Communication-TimeOut`	`public int CommunicationTimeOut {get; set;}`	Get or set the communication timeout in seconds.	`xPCException` — When problem occurs, query xPCException object `Reason` property.
`Component`	`public IComponent Component {get;}`	Get component associated with the ISite when implemented by a class.	
`Container`	`public IContainer Container {get;}`	Get the IContainer associated with the ISite when implemented by a class.	
`Container-Control`	`public ContainerControl ContainerControl {get; set;}`	Provide focus-management functionality for controls that can function as containers for other controls.	
`DLMFileName`	`public string DLMFileName {get; set;}`	Get or set the full path to the DLM file name.	
`Echo`	`public bool Echo {get; set;}`	Get or set the target display on the target computer.	`xPCException` — When problem occurs, query xPCException object `Reason` property.

Property	C# Declaration Syntax	Description	Exception
FileSystem	public xPCFileSystem FileSystem {get;}	Get a reference to an xPCFileSystem object that you can use to interface with the target file system. If no communication is established, the property returns a NULL object.	
HostTarget-Comm	public XPCProtocol HostTargetComm {get; set;}	Get or set the physical medium for communication. See xPCProtocol Enumerated Data Type.	
IsConnected	public bool IsConnected {get;}	Get connection status (established or not) to a remote target computer.	
IsConnecting-Busy	public bool IsConnectingBusy {get;}	Get ConnectAsync request status (in progress or not).	
IsDiscon-nectingBusy	public bool IsDisconnectingBusy {get;}	Get whether a DisconnectAsync request is in progress.	
IsLoadingBusy	public bool IsLoadingBusy {get;}	Gets LoadAsync request status (in progress or not).	
IsRebooting-Busy	public bool IsRebootingBusy {get;}	Get RebootAsync request status (in progress or not).	
IsUnloading-Busy	public bool IsUnloadingBusy {get;}	Gets unLoadingAsync request status (in progress or not).	
RS232BaudRate	public XPCRS232BaudRate RS232Baudrate {get; set;}	Get or set baudrate for serial link. See xPCRS232BaudRate Enumerated Data Type.	

Property	C# Declaration Syntax	Description	Exception
RS232HostPort	public XPCRS232CommPort RS232HostPort {get; set;}	Get or set the serial COM port for connection on development computer. The Simulink Real-Time software automatically determines the COM port on the target computer. See xPCRS232Comport Enumerated Data Type.	
SessionTime	public double SessionTime {get;}	Get the length of time Simulink Real-Time kernel has been running on the target computer.	xPCException — When problem occurs, query xPCException object Reason property.
Site	public ISite Site {get; set;}	Get or set site of the control.	
TargetPCName	public string TargetPCName {get; set;}	Get or set a value indicating the target computer name associated with the target computer.	
TcpIpTarget-Address	public string TcpIpTargetAddress {get; set;}	Get or set a valid IP address for your target computer.	
TcpIpTarget-Port	public string TcpIpTargetPort {get; set;}	Get or set the TCP/IP target port. The default is 22222 and should not cause problems. This number is higher than the reserved area (for example, the port numbers reserved for telnet or ftp). The software uses this value only for the target computer.	

xPCTargetScope Class

Access to target scopes

Syntax

```
public class xPCTargetScope : xPCScope
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class xPCTargetScope : xPCScope` initializes a new instance of the xPCTargetScope class.

Methods

The xPCTargetScope class inherits methods from `xPCScope Class`.

Events

The xPCTargetScope class inherits events from `xPCScope Class`.

Properties

The xPCTargetScope class inherits its other properties from `xPCScope Class`.

Property	C# Declaration Syntax	Description	Exception
`DisplayMode`	`public SCDISPLAYMODE DisplayMode {get; set;}`	Get or set scope mode for displaying signals.	`xPCException` — When problem occurs, query xPCException object `Reason` property.

Property	C# Declaration Syntax	Description	Exception
Grid	public bool Grid {get; set;}	Get or set status of grid line for particular scope.	xPCException — When problem occurs, query xPCException object Reason property.
Signals	public xPCTargetScope-SignalCollection Signals {get;}	Get the collection of target scope signals xPCTarget-ScopeSignalCollection that you assign to this scope object.	
Trigger-Signal	public xPCTgtScopeSignal TriggerSignal {get; set;}	Get or set target scope signal xPCTgtScopeSignal used to trigger the scope.	xPCException — When problem occurs, query xPCException object Reason property.
YLimit	public double[] YLimit {get; set;}	Get or set *y*-axis minimum and maximum limits for scope.	xPCException — When problem occurs, query xPCException object Reason property.

xPCTargetScopeCollection Class

Collection of `xPCTargetScope` objects

Syntax

```
public class xPCTargetScopeCollection :
xPCScopeCollection<xPCTargetScope>
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class xPCTargetScopeCollection : xPCScopeCollection<xPCTargetScope>` initializes collection of xPCTargetScope objects.

Methods

Method	Description
xPCTargetScopeCollection.Ac	Create xPCTargetScope object with the next available scope ID as key
xPCTargetScopeCollection.Re	Refresh target scope object state
xPCTargetScopeCollection.St	Start all target scopes in one call
xPCTargetScopeCollection.St	Stop all target scopes in one call

xPCTargetScopeSignalCollection Class

Collection of `xPCHostScopeSignal` objects

Syntax

```
public class xPCTargetScopeSignalCollection :
xPCScopeSignalCollection
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

```
public class xPCTargetScopeSignalCollection :
xPCScopeSignalCollection.
```

Methods

Method	Description
xPCTargetScopeSignalCollect	Create xPCTargetScopeSignal object
xPCTargetScopeSignalCollect	Synchronize signals for associated target scopes on target computer

Properties

Property	C# Declaration Syntax	Description	Exception
`Item`	`public xPCTgtScopeSignal Item[ string blkpath ] {get;}`	Get xPCTgtScopeSignal object from signal name (*`blkpath`*). *`blkpath`* is the signal name that represents a	`xPCException` — When problem occurs, query xPCException object `Reason` property.

Property	C# Declaration Syntax	Description	Exception
		signal object added to its parent xPCTargetScope object. This property returns the file scope signal object as type xPCTgtScopeSignal.	

xPCTETLogger Class

Access to task execution time (TET) logger

Syntax

```
public class xPCTETLogger : xPCLog
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class xPCTETLogger : xPCLog` initializes a new instance of the xPCTETLogger class.

Properties

The xPCTETLogger class inherits its other properties from `xPCLog Class`.

Properties	C# Declaration Syntax	Description
`DataLogObject`	`public xPCDataLoggingObject DataLogObject {get;}`	Get TET data logging object.
`IsEnabled`	`public override bool IsEnabled {get;}`	Get whether to enable or disable logging. Overrides xPCLog.IsEnabled.

xPCTgtScopeSignal Class

Access to target scope signals

Syntax

```
public class xPCTgtScopeSignal : xPCScopeSignal
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class xPCTgtScopeSignal : xPCScopeSignal` initializes access to target scope signals.

Properties

Property	C# Declaration Syntax	Description	Exception
`Numerical Format`	`public string NumericalFormat {get; set;}`	Get and set numerical format for the numeric displayed signal associated with this object.	`xPCException` — When problem occurs, query xPCException object `Reason` property.
`Scope`	`public xPCTargetScope Scope {get;}`	Get parent target scope xPCTargetScope object.	

xPCTimeLogger Class

Access to output log

Syntax

```
public class xPCTimeLogger : xPCLog
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class xPCTimeLogger : xPCLog` initializes a new instance of the xPCTimeLogger class.

Properties

The xPCTimeLogger class inherits its other properties from `xPCLog Class`.

Properties	C# Declaration Syntax	Description
`DataLogObjects`	`public xPCDataLoggingObject DataLogObject {get;}`	Get the xPCDataLoggingObject of the time log.
`IsEnabled`	`public override bool IsEnabled {get;}`	Get whether to enable or disable logging. Overrides xPCLog.IsEnabled.

xPCFileInfo.Open

Open file

Syntax

```
public xPCFileStream Open(xPCFileMode fileMode)
```

Description

Class: `xPCFileInfo Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public xPCFileStream Open(xPCFileMode fileMode)` opens file with specified mode. This method returns the xPCFileStream object for the file. See xPCFileMode Enumerated Data Type for file mode options.

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCFileInfo.OpenRead

Create read-only `xPCFileStream` object

Syntax

```
public xPCFileStream OpenRead()
```

Description

Class: `xPCFileInfo Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public xPCFileStream OpenRead()` creates a read-only xPCFileStream object. This method returns the xPCFileStream object for the file.

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCTargetPC.Ping

Test communication between development and target computers

Syntax

```
public bool Ping()
```

Description

Class: `xPCTargetPC Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public bool Ping()` tests the communication between development and target computers. This method returns a Boolean value.

xPCFileStream.Read

Read block of bytes from stream and write data to buffer

Syntax

```
public int Read(byte[] buffer, int offset, int count)
```

Description

Class: `xPCFileStream Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public int Read(byte[] buffer, int offset, int count)` reads a block of bytes from the file stream. It then writes the data to the specified buffer, *`buffer`*. *`buffer`* specifies the size in bytes and is a `byte` structure (8-bit unsigned integer). When this method returns, it contains the byte array with the values between *`offset`* and (*`offset`* + *`count`* - 1), replaced by the bytes read from the current source. *`offset`* is an integer. It specifies the byte offset in the array at which the method places the read bytes. *`count`* is an integer. It specifies the number of bytes to read from the stream. This method returns the total number of bytes the method reads into the buffer. This number might be less than the number of bytes requested if that number of bytes are not currently available. It can also be zero if the method reaches the end of the stream.

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCTargetPC.Reboot

Restart target computer

Syntax

```
public void Reboot()
```

Description

Class: `xPCTargetPC Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void Reboot()` restarts the target computer.

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCTargetPC.RebootAsync

Asynchronous request to restart target computer

Syntax

```
public void RebootAsync()
```

Description

Class: `xPCTargetPC Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void RebootAsync()` begins an asynchronous request to restart a target computer.

Exception

Exception	Condition
`InvalidOperation-Exception`	When another thread uses this method.

xPCTargetPC.RebootCompleted

Event when `xPCTargetPC.RebootAsync` is complete

Syntax

```
public event RebootCompletedEventHandler RebootCompleted
```

Description

Class: `xPCTargetPC Class`

Event

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public event RebootCompletedEventHandler RebootCompleted` occurs when an asynchronous restart operation is complete.

xPCTargetPC.Rebooted

Event after `xPCTargetPC.Reboot` is complete

Syntax

```
public event EventHandler Rebooted
```

Description

Class: `xPCTargetPC Class`

Event

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public event EventHandler Rebooted` occurs after a target computer restart is complete.

xPCTargetPC.Rebooting

Event before `xPCTargetPC.Reboot` starts

Syntax

```
public event EventHandler Rebooting
```

Description

Class: `xPCTargetPC Class`

Event

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public event EventHandler Rebooting` occurs before a restart operation executes.

xPCFileScopeCollection.Refresh

Synchronize with file scopes on target computer

Syntax

```
public override void Refresh()
```

Description

Class: `xPCFileScopeCollection Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public override void Refresh()` synchronizes with file scopes on target computer.

Overrides xPCScopeCollection<xPCFileScope>.Refresh().

xPCScopes.RefreshAll

Refresh state of object

Syntax

```
public void RefreshAll()
```

Description

Class: `xPCScopes Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void RefreshAll()` refreshes state of object.

xPCDriveInfo.Refresh

Synchronize with file drives on target computer

Syntax

```
public void Refresh()
```

Description

Class: `xPCDriveInfo Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void Refresh()` synchronizes with file drives on target computer.

xPCFileScopeSignalCollection.Refresh

Synchronize with signals for associated scope on target computer

Syntax

```
public override void Refresh()
```

Description

Class: `xPCFileScopeSignalCollection Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public override void Refresh()` synchronizes with signals for associated file scopes on target computer.

Overrides xPCScopeCollection<xPCFileScopeSignal>.Refresh().

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCHostScopeCollection.Refresh

Refresh host scope object state

Syntax

```
public override void Refresh()
```

Description

Class: `xPCHostScopeCollection Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public override void Refresh()` refreshes host scope object state.

Overrides xPCScopeCollection<xPCHostScope>.Refresh().

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCHostScopeSignalCollection.Refresh

Synchronize signals for associated host scopes on target computer

Syntax

```
public override void Refresh()
```

Description

Class: `xPCHostScopeSignalCollection Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public override void Refresh()` synchronizes signals for associated host scopes on target computer.

Overrides xPCScopeCollection<xPCHostScope>.Refresh().

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCParameters.Refresh

Refresh state of object

Syntax

```
public override void Refresh()
```

Description

Class: `xPCParameters Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public override void Refresh()` refreshes the state of the object.

xPCSignals.Refresh

Refresh state of object

Syntax

```
public void Refresh()
```

Description

Class: `xPCSignals Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void Refresh()` refreshes the state of the object.

xPCTargetScopeCollection.Refresh

Refresh target scope object state

Syntax

```
public override void Refresh()
```

Description

Class: `xPCTargetScopeCollection Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public override void Refresh()` refreshes target scope object state.

Overrides xPCScopeCollection<xPCTargetScope>.Refresh().

xPCTargetScopeSignalCollection.Refresh

Synchronize signals for associated target scopes on target computer

Syntax

```
public override void Refresh()
```

Description

Class: `xPCTargetScopeSignalCollection Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public override void Refresh()` synchronizes signals for associated target scopes on target computer.

Overrides xPCScopeSignalCollection<xPCTgtScopeSignal>.Refresh().

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCFileSystem.RemoveFile

Remove file name from target computer

Syntax

```
public void RemoveFile(string fileName)
```

Description

Class: `xPCFileSystem Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void RemoveFile(string fileName)` removes the specified file name from the target computer. *`fileName`* is a string that specifies the full path name to the file you want to remove.

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCFileInfo.Rename

Rename file

Syntax

```
public xPCFileInfo Rename(string newName)
```

Description

Class: `xPCFileInfo Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public xPCFileInfo Rename(string newName)` changes file name to *`newName`*. *`newName`* is a string. This method returns the xPCFileInfo object.

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCParameters.SaveParameterSet

Save parameter values of real-time application

Syntax

```
public void SaveParameterSet(string fileName)
```

Description

Class: `xPCParameters Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void SaveParameterSet(string fileName)` saves parameter values of the real-time application in a file. *`fileName`* is a string that represents the file to contain the saved parameter values.

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

SCDISPLAYMODE Enumerated Data Type

Target scope display mode values

Syntax

```
public enum SCDISPLAYMODE
```

Description

Enumerated Data Type

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public enum SCDISPLAYMODE` specifies target scope display mode values.

Members

Member	Description
`NUMERICAL`	Specifies target scope drawing mode to display numerical value.
`REDRAW`	Specifies target scope drawing mode to redraw mode.
`SLIDING`	Specifies target scope drawing mode to sliding mode.
`ROLLING`	Specifies target scope drawing mode to rolling mode.

SCFILEMODE Enumerated Data Type

Write mode values for when file allocation table entry is updated

Syntax

```
public enum SCFILEMODE
```

Description

Enumerated Data Type

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public enum SCFILEMODE` specifies write mode values for when file allocation table entry is updated.

Members

Member	Description
`LAZY`	Enables lazy write mode.
`COMMIT`	Enables commit write mode.

xPCScope.ScopeStarted

Event after `xPCScope.Start` is complete

Syntax

```
public event EventHandler ScopeStarted
```

Description

Class: `xPCScope Class`

Event

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public event EventHandler ScopeStarted` occurs after a scope start command is complete.

xPCScope.ScopeStarting

Event before `xPCScope.Start` executes

Syntax

```
public event EventHandler ScopeStarting
```

Description

Class: `xPCScope Class`

Event

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public event EventHandler ScopeStarting` occurs before a scope executes.

xPCScope.ScopeStopped

Event after `xPCScope.Stop` is complete

Syntax

```
public event EventHandler ScopeStarting
```

Description

Class: `xPCScope Class`

Event

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public event EventHandler ScopeStarting` occurs after a scope completes a manual stop command.

xPCScope.ScopeStopping

Event before `xPCScope.Stop` executes

Syntax

```
public event EventHandler ScopeStopping
```

Description

Class: `xPCScope Class`

Event

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public event EventHandler ScopeStopping` occurs before a scope completes a manual stop.

SCSTATUS Enumerated Data Type

Scope status values

Syntax

```
public enum SCSTATUS
```

Description

Enumerated Data Type

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public enum SCSTATUS` specifies scope status values.

Members

Member	Description
`WAITTOSTART`	Scope is ready and waiting to start.
`WAITFORTRIG`	Scope is finished with the preacquiring state and waiting for a trigger. If the scope does not preacquire data, it enters the wait for trigger state.
`ACQUIRING`	Scope is acquiring data. The scope enters this state when it leaves the wait for trigger state.
`FINISHED`	Scope is finished acquiring data when it has attained the predefined limit.
`INTERRUPTED`	The user has stopped (interrupted) the scope.
`PREACQUIRING`	Scope acquires a predefined number of samples before triggering.

SCTRIGGERMODE Enumerated Data Type

Scope trigger mode values

Syntax

```
public enum SCTRIGGERMODE
```

Description

Enumerated Data Type

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public enum SCTRIGGERMODE` specifies scope trigger mode values.

Members

Member	Description
`FREERUN`	There is no external trigger condition.. The scope triggers when it is ready to trigger, regardless of the circumstances.
`SOFTWARE`	Only user intervention can trigger the scope, and it can do so regardless of circumstances. No other triggering is possible.
`SIGNAL`	Signal must cross a value before the scope is triggered.
`SCOPE`	Scope is triggered by another scope at a predefined trigger point of the triggering scope. You modify this trigger point with the value of `TriggerScopeSample`.

SCTRIGGERSLOPE Enumerated Data Type

Scope trigger slope values

Syntax

```
public enum SCTRIGGERSLOPE
```

Description

Enumerated Data Type

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public enum SCTRIGGERSLOPE` specifies scope trigger slope values.

Members

Member	Description
`EITHER`	The trigger slope can be rising or falling.
`RISING`	The trigger signal value must be rising when it crosses the trigger value.
`FALLING`	The trigger signal value must be falling when it crosses the trigger value.

SCTYPE Enumerated Data Type

Scope type

Syntax

```
public enum SCTYPE
```

Description

Enumerated Data Type

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public enum SCTYPE` specifies scope type.

Members

Member	Description
`HOST`	Specifies scope as type host.
`TARGET`	Specifies scope as type target.
`FILE`	Specifies scope as type file.

xPCFileSystem.SetCurrentDirectory

Current folder

Syntax

```
public void SetCurrentDirectory(string path)
```

Description

Class: `xPCFileSystem Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void SetCurrentDirectory(string path)` sets the current folder to the specified path name on the target computer. *`path`* is a string that specifies the full path name to the folder you want to make current.

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCParameter.SetParam

Change value of parameter on target computer

Syntax

```
public void SetParam(double[] values)
```

Description

Class: `xPCParameter Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void SetParam(double[] values)` sets the parameter to *`values`*. Parameter *`values`* is a vector of doubles, assumed to be the size required by the parameter type.

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCParameter.SetParamAsync

Asynchronous request to change parameter value on target computer

Syntax

```
public void SetParamAsync(double[] values)
public void SetParamAsync(double[] values, Object taskId)
```

Description

Class: `xPCParameter Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void SetParamAsync(double[] values)` begins an asynchronous request to set parameter values to *`values`* on the target computer. This method does not block the calling thread. *`values`* is a vector of double values to which to set the parameter values.

`public void SetParamAsync(double[] values, Object taskId)` receives a user-defined object when it completes its asynchronous request. *`values`* is a vector of double values to which to set the parameter values. *`taskId`* is a user-defined object that you can have passed to the `SetParamAsync` method upon completion.

Exception

Exception	Condition
`InvalidOperation-Exception`	When another thread uses this method.

xPCParameter.SetParamCompleted

Event when `xPCParameter.SetParamAsync` is complete

Description

Class: `xPCParameter Class`

Event

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public event SetParamCompletedEventHandler SetParamCompleted` occurs when an asynchronous set parameter operation is complete.

xPCApplication.Start

Start real-time application execution

Syntax

```
public void Start()
```

Description

Class: `xPCApplication Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void Start()` starts the real-time application simulation.

Exception

Exception	Condition
`xPCException`	When problem occurs, query xPCException object `Reason` property.

xPCFileScopeCollection.StartAll

Start all file scopes in one call

Syntax

```
public void StartAll()
```

Description

Class: `xPCFileScopeCollection Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void StartAll()` sequentially starts all file scopes using one call. This method starts all the file scopes in the xPCFileScopeCollection.

xPCHostScopeCollection.StartAll

Start all host scopes in one call

Syntax

```
public void StartAll()
```

Description

Class: `xPCHostScopeCollection Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void StartAll()` sequentially starts all host scopes using one call. This method starts all the host scopes in the xPCHostScopeCollection.

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCTargetScopeCollection.StartAll

Start all target scopes in one call

Syntax

```
public void StartAll()
```

Description

Class: `xPCTargetScopeCollection Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void StartAll()` sequentially starts all target scopes using one call. This method starts all the target scopes in the xPCTargetScopeCollection.

xPCScope.Start

Start scope

Syntax

```
public void Start()
```

Description

Class: `xPCScope Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void Start()` starts execution of scope on target computer.

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCApplication.Started

Event after `xPCApplication.Start` is complete

Syntax

```
public event EventHandler Started
```

Description

Class: `xPCApplication Class`

Event

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public event EventHandler Started` occurs after a real-time application start command is complete.

xPCApplication.Starting

Event before `xPCApplication.Start` executes

Syntax

```
public event EventHandler Starting
```

Description

Class: `xPCApplication Class`

Event

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public event EventHandler Starting` occurs before a real-time application start command executes.

xPCApplication.Stop

Stop real-time application execution

Syntax

```
public void Stop()
```

Description

Class: `xPCApplication Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void Stop()` stops the real-time application simulation.

Exception

Exception	Condition
`xPCException`	When problem occurs, query xPCException object `Reason` property.

xPCFileScopeCollection.StopAll

Stop all file scopes in one call

Syntax

```
public void StopAll()
```

Description

Class: `xPCFileScopeCollection Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void StopAll()` stops all file scopes using one call. This method stops all the file scopes in the xPCFileScopeCollection.

xPCHostScopeCollection.StopAll

Stop all host scopes in one call

Syntax

```
public void StopAll()
```

Description

Class: `xPCHostScopeCollection Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void StopAll()` sequentially stops all host scopes using one call. This method stops all the host scopes in the xPCHostScopeCollection.

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCTargetScopeCollection.StopAll

Stop all target scopes in one call

Syntax

```
public void StopAll()
```

Description

Class: `xPCTargetScopeCollection Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void StopAll()` sequentially stops all target scopes using one call. This method stops all the target scopes in the xPCTargetScopeCollection.

xPCScope.Stop

Stop scope

Syntax

```
public void Stop()
```

Description

Class: `xPCScope Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void Stop()` stops execution of scope on target computer.

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCApplication.Stopped

Event after `xPCApplication.Stop` is complete

Syntax

```
public event EventHandler Stopped
```

Description

Class: `xPCApplication Class`

Event

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public event EventHandler Stopped` occurs after a real-time application stop command is complete.

xPCApplication.Stopping

Event before `xPCApplication.Stop` executes

Syntax

```
public event EventHandler Stopping
```

Description

Class: `xPCApplication Class`

Event

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public event EventHandler Stopping` occurs before a real-time application stop command executes.

xPCTargetPC.tcpPing

Determine TCP/IP accessibility of remote computer

Syntax

```
public bool tcpPing()
```

Description

Class: `xPCTargetPC Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public bool tcpPing()` allows a real-time application to determine whether a remote computer is accessible on the TCP/IP network. This method returns a Boolean value.

xPCScope.Trigger

Software-trigger start of data acquisition for scope

Syntax

```
public void Trigger()
```

Description

Class: `xPCScope Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void Trigger()` software-triggers start of data acquisition for current scope.

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCSignal.TryGetValue

Status of get signal value at moment of request

Syntax

```
public virtual bool TryGetValue(ref double result)
```

Description

Class: `xPCSignal Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public virtual bool TryGetValue(ref double result)` returns the status of get signal value at moment of request. If the software detects an error, this method returns false. Otherwise, the method returns true.

xPCTargetPC.Unload

Unload real-time application from target computer

Syntax

```
public void Unload()
```

Description

Class: `xPCTargetPC Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void Unload()` unloads a real-time application from a target computer.

Exception

Exception	Condition
`xPCException`	When problem occurs, query xPCException object `Reason` property.

xPCTargetPC.UnloadAsync

Asynchronous request to unload real-time application from target computer

Syntax

```
public void UnloadAsync()
```

Description

Class: `xPCTargetPC Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void UnloadAsync()` begins an asynchronous request to unload a real-time application from a target computer.

Exception

Exception	Condition
`InvalidOperation-Exception`	When another thread uses this method.

xPCTargetPC.UnloadCompleted

Event when `xPCTargetPC.UnloadAsync` is complete

Syntax

```
public event UnloadCompletedEventHandler UnloadCompleted
```

Description

Class: `xPCTargetPC Class`

Event

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public event UnloadCompletedEventHandler UnloadCompleted` occurs when an asynchronous real-time application unload operation is complete.

xPCTargetPC.Unloaded

Event after `xPCTargetPC.Unload` is complete

Syntax

```
public event EventHandler Unloaded
```

Description

Class: `xPCTargetPC Class`

Event

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public event EventHandler Unloaded` occurs after a real-time application unload from the target computer is complete.

xPCTargetPC.Unloading

Event before `xPCTargetPC.Unload` starts

Syntax

```
public event EventHandler Unloading
```

Description

Class: `xPCTargetPC Class`

Event

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public event EventHandler Unloading` occurs before a real-time application unload from a target computer starts.

xPCFileStream.Write

Write block of bytes to file stream

Syntax

```
public void Write(byte[] buffer, int count)
```

Description

Class: `xPCFileStream Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void Write(byte[] buffer, int count)` writes data from a block of bytes, *`buffer`*, to the current file stream. *`buffer`* contains the data to write to the stream. It is a `byte` structure. *`count`* is an integer. It specifies the number of bytes to write to the current file stream.

Exception

Exception	Condition
`xPCException`	When problem occurs, query xPCException object `Reason` property.

xPCFileStream.WriteByte

Write byte to current position in file stream

Syntax

```
public void WriteByte(byte value)
```

Description

Class: `xPCFileStream Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public void WriteByte(byte value)` writes a byte to the current position in the file stream. *`value`* contains the byte of data that the method writes to the file stream.

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCAppStatus Enumerated Data Type

Real-time application status return values

Syntax

```
public enum xPCAppStatus
```

Description

Enumerated Data Type

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public enum xPCAppStatus` specifies real-time application status return values.

Members

Member	Description
`Stopped`	Real-time application is stopped
`Running`	Real-time application is running

xPCDirectoryInfo

Construct new instance of `xPCDirectoryInfo` class on specified path

Syntax

```
public xPCDirectoryInfo(xPCTargetPC tgt, string path)
```

Description

Class: `xPCDirectoryInfo Class`

Constructor

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public xPCDirectoryInfo(xPCTargetPC tgt, string path)` initializes a new instance of the xPCDirectoryInfo class on the path, *`path`*. *`tgt`* is an xPCTargetPC object that represents the target computer for which you initialize the class. *`path`* is a string that represents the path on which to create the xPCDirectoryInfo object.

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCDriveInfo

Construct new instance of `xPCDriveInfo` class

Syntax

```
public xPCDriveInfo(xPCTargetPC tgt, string driveName)
```

Description

Class: `xPCDriveInfo Class`

Constructor

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public xPCDriveInfo(xPCTargetPC tgt, string driveName)` initializes a new instance of the xPCDriveInfo class. *`tgt`* is an xPCTargetPC object that represents the target computer for which you want to the return drive information. *`driveName`* is a string that represents the name of the drive.

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCException

Construct new instance of `xPCException` class

Syntax

```
public xPCException()
public xPCException(string message)
public xPCException(string message, Exception inner)
public xPCException(SerializationInfo info, StreamingContext
context)
public xPCException(int errId, string message, xPCTargetPC tgt)
```

Description

Class: `xPCException Class`

Constructor

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public xPCException()` initializes a new instance of the xPCException class.

`public xPCException(string message)` initializes a new instance of the xPCException class with *`message`*. *`message`* is a string that contains the text of the error message.

`public xPCException(string message, Exception inner)` initializes a new instance of the xPCException class with *`message`* and *`inner`*. *`message`* is a string. *`inner`* is a nested Exception object.

`public xPCException(SerializationInfo info, StreamingContext context)` initializes a new instance of the xPCException class with serialization information, *`info`*, and streaming context, *`context`*. *`info`* is a SerializationInfo object. *`context`* is a StreamingContext object.

`public xPCException(int errId, string message, xPCTargetPC tgt)` initializes a new instance of the xPCException class. *`errID`* is a 32–bit integer that contains the error ID numbers as defined in *`matlabroot`*`\toolbox\rtw\targets\xpc\api\xpcapiconst.h`. *`message`* is an error message string. *`tgt`* is the xPCTargetPC object that raised the error.

xPCExceptionReason Enumerated Data Type

Exception reasons

Syntax

```
public enum xPCExceptionReason
```

Description

Enumerated Data Type

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public enum xPCExceptionReason` specifies the reasons for an exception. See "C API Error Messages" for definitions.

xPCFileInfo

Construct new instance of `xPCFileInfo` class

Syntax

```
public xPCFileInfo(xPCTargetPC tgt, string fileName)
```

Description

Class: `xPCFileInfo Class`

Constructor

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public xPCFileInfo(xPCTargetPC tgt, string fileName)` initializes a new instance of the xPCFileInfo class. *`tgt`* is an xPCTargetPC object that represents the target computer for which you want to return the file information. *`fileName`* is a string that represents the name of the file. It is a fully qualified name of the new file, or the relative file name in the target computer file system.

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCFileMode Enumerated Data Type

Open file with permissions

Syntax

```
public enum xPCFileMode
```

Description

Enumerated Data Type

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public enum xPCFileMode` specifies how the target computer is to open a file with permissions.

Members

Member	Description
`CreateWrite`	Open file for writing and discard existing contents.
`CreateReadWrite`	Open or create file for reading and writing and discard existing contents
`OpenRead`	Open file for reading
`OpenReadWrite`	Open (but do not create) file for reading and writing
`AppendWrite`	Open or create file for writing and append data to end of file
`AppendReadWrite`	Open or create file for reading and writing and append data to end of file

xPCFileStream

Construct new instance of `xPCFileStream` class

Syntax

```
public xPCFileStream(xPCTargetPC tgt, string path, xPCFileMode
fmode)
```

Description

Class: `xPCFileStream Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public xPCFileStream(xPCTargetPC tgt, string path, xPCFileMode fmode)` initializes a new instance of the xPCFileStream class with the path name and creation mode. *`tgt`* is a reference to an xPCTargetPC object. *`path`* is a relative or absolute path name for the file that the current xPCFileStream object encapsulates. *`fmode`* is an xPCFileMode constant that determines how to open or create the file. See xPCFileMode Enumerated Data Type for file mode options.

Exception

Exception	Condition
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

xPCFileSystemInfo

Construct new instance of `xPCFileSystemInfo` class

Syntax

```
public xPCFileSystemInfo(xPCTargetPC tgt)
```

Description

Class: `xPCFileSystemInfo Class`

Constructor

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public xPCFileSystemInfo(xPCTargetPC tgt)` initializes a new instance of the xPCFileSystemInfo class. *`tgt`* is an xPCTargetPC object that represents the target computer for which you want the file system information.

xPCLogMode Enumerated Data Type

Specify log mode values

Syntax

```
public enum xPCLogMode
```

Description

Enumerated Data Type

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public enum xPCLogMode` specifies log mode values.

Members

Member	Description
`Normal`	Time-equidistant logging to log data point at every time interval.
`Value`	Log data point only when output signal from OutputLog increments by a specified value

xPCLogType Enumerated Data Type

Logging type values

Syntax

```
public enum xPCLogType
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Enumerated Data Type

Syntax Language: C#

`public enum xPCLogType` specifies logging type values.

Members

Member	Description
`OUTPUTLOG`	Output log
`STATELOG`	State log
`TIMELOG`	Time log
`TETLOG`	TET log

xPCProtocol Enumerated Data Type

Development computer and target computer communication medium

Syntax

```
public enum XPCProtocol
```

Description

Enumerated Data Type

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public enum XPCProtocol` specifies development computer and target computer communication medium.

Members

Member	Description
`RS232`	Serial link **Note:** `RS232` communication type will be removed in a future release. Use `TCPIP` instead.
`TCPIP`	Ethernet link

xPCRS232BaudRate Enumerated Data Type

Serial link baud rate

Syntax

```
public enum XPCRS232BaudRate
```

Description

Enumerated Data Type

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public enum XPCRS232BaudRate` specifies serial link baud rate

Members

Member	Description
`BAUD1200`	1200 baud rate
`BAUD2400`	2400 baud rate
`BAUD4800`	4800 baud rate
`BAUD9600`	9600 baud rate
`BAUD19200`	19200 baud rate
`BAUD38400`	38400 baud rate
`BAUD57600`	57600 baud rate
`BAUD115200`	115200 baud rate

Note: RS-232 communication type will be removed in a future release. Use TCP/IP instead.

xPCRS232Comport Enumerated Data Type

Serial link port

Syntax

```
public enum XPCRS232CommPort
```

Description

Enumerated Data Type

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public enum XPCRS232CommPort` specifies values of the supported serial link ports used for the connection on the development computer.

Members

Member	Description
`COM1`	Serial port COM 0
`COM2`	Serial port COM 1

Note: RS-232 communication type will be removed in a future release. Use TCP/IP instead.

xPCTargetPC

Construct new instance of `xPCTargetPC` class

Syntax

```
public xPCTargetPC()
```

Description

Class: `xPCTargetPC Class`

Constructor

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public xPCTargetPC()` initializes a new instance of the xPCTargetPC class.

Simulink Real-Time API Reference for C

dirStruct

Type definition for file system folder information structure

Syntax

```
typedef struct {
   char         Name[8];
   char         Ext[3];
   char         Day;
   int  Month;
   int  Year;
   int  Hour;
   int  Min;
   int  isDir;
   unsigned long  Size;
} dirStruct;
```

Fields

Name
This value contains the name of the file or folder.

Ext
This value contains the file type of the element, if the element is a file (*isDir* is 0). If the element is a folder (*isDir* is 1), this field is empty.

Day
This value contains the day the file or folder was last modified.

Month
This value contains the month the file or folder was last modified.

Year
This value contains the year the file or folder was last modified.

Hour
This value contains the hour the file or folder was last modified.

Min
This value contains the minute the file or folder was last modified.

isDir
This value indicates if the element is a file (0) or folder (1). If it is a folder, Bytes has a value of 0.

`Size`	This value contains the size of the file in bytes. If the element is a folder, this value is `0`.

Description

The `dirStruct` structure contains information for a folder in the file system.

See Also

API function `xPCFSDirItems`

diskinfo

Type definition for file system disk information structure

Syntax

```
typedef struct {
   char          Label[12];
   char          DriveLetter;
   char          Reserved[3];
   unsigned int  SerialNumber;
   unsigned int  FirstPhysicalSector;
   unsigned int  FATType;
   unsigned int  FATCount;
   unsigned int  MaxDirEntries;
   unsigned int  BytesPerSector;
   unsigned int  SectorsPerCluster;
   unsigned int  TotalClusters;
   unsigned int  BadClusters;
   unsigned int  FreeClusters;
   unsigned int  Files;
   unsigned int  FileChains;
   unsigned int  FreeChains;
   unsigned int  LargestFreeChain;
} diskinfo;
```

Fields

Label — This value contains the zero-terminated string that contains the volume label. The string is empty if the volume has no label.

DriveLetter — This value contains the drive letter, in uppercase.

Reserved — Reserved.

SerialNumber — This value contains the volume serial number.

FirstPhysicalSector — This value contains the logical block addressing (LBA) address of the logical drive boot record. For 3.5-inch disks, this value is `0`.

FATType This value contains the type of file system found. It can contain 12 , 16 , or 32 for FAT-12, FAT-16, or FAT-32 volumes, respectively.

FATCount This value contains the number of FAT partitions on the volume.

MaxDirEntries This value contains the size of the root folder. For FAT-32 systems, this value is 0.

BytesPerSector This value contains the sector size. This value is most likely to be `512`.

SectorsPerCluster This value contains, in sectors, the size of the smallest unit of storage that can be allocated to a file.

TotalClusters This value contains the number of file storage clusters on the volume.

BadClusters This value contains the number of clusters that have been marked as bad. These clusters are unavailable for file storage.

FreeClusters This value contains the number of clusters that are currently available for storage.

Files This value contains the number of files, including folders, on the volume. This number excludes the root folder and files that have an allocated file size of 0.

FileChains This value contains the number of contiguous cluster chains. On a completely unfragmented volume, this value is identical to the value of `Files`.

FreeChains This value contains the number of contiguous cluster chains of free clusters. On a completely unfragmented volume, this value is 1.

LargestFreeChain This value contains the maximum allocated file size, in number of clusters, for a newly allocated contiguous file. On a completely unfragmented volume, this value is identical to `FreeClusters`.

Description

The `diskinfo` structure contains information for file system disks.

See Also

API function `xPCFSDiskInfo`

fileinfo

Type definition for file information structure

Syntax

```
typedef struct {
int           FilePos;
int           AllocatedSize;
int           ClusterChains;
int           VolumeSerialNumber;
char          FullName[255];
}fileinfo;
```

Fields

FilePos This value contains the current file pointer.

AllocatedSize This value contains the currently allocated file size.

ClusterChains This value indicates how many separate cluster chains are allocated for the file.

VolumeSerialNumber This value holds the serial number of the volume the file resides on.

FullName This value contains a copy of the complete path name of the file. This field is valid only while the file is open.

Description

The `fileinfo` structure contains information for files in the file system.

See Also

`xPCFSFileInfo`

lgmode

Type definition for logging options structure

Syntax

```
typedef struct {
   int    mode;
   double incrementvalue;
} lgmode;
```

Fields

mode

This value indicates the type of logging you want. Specify LGMOD_TIME for time-equidistant logging. Specify LGMOD_VALUE for value-equidistant logging.

incrementvalue

If you set *mode* to LGMOD_VALUE for value-equidistant data, this option specifies the increment (difference in amplitude) value between logged data points. A data point is logged only when an output signal or a state changes by *incrementvalue*.

If you set *mode* to LGMOD_TIME, *incrementvalue* is ignored.

Description

The lgmode structure specifies data logging options. The *mode* variable accepts either the numeric values 0 or 1 or their equivalent constants LGMOD_TIME or LGMOD_VALUE from xpcapiconst.h.

See Also

API functions xPCSetLogMode, xPCGetLogMode

scopedata

Type definition for scope data structure

Syntax

```
typedef struct {
   int    number;
   int    type;
   int    state;
   int    signals[10];
   int    numsamples;
   int    decimation;
   int    triggermode;
   int    numprepostsamples;
   int    triggersignal
   int    triggerscope;
   int    triggerscopesample;
   double triggerlevel;
   int    triggerslope;
} scopedata;
```

Fields

number

The scope number.

type

Determines whether the scope is displayed on the development computer or on the target computer. Values are one of the following:

1	Host
2	Target

state

Indicates the scope state. Values are one of the following:

0	Waiting to start
1	Scope is waiting for a trigger
2	Data is being acquired
3	Acquisition is finished
4	Scope is stopped (interrupted)

5 Scope is preacquiring data

signals
List of signal indices from the target object to display on the scope.

numsamples
Number of contiguous samples captured during the acquisition of a data package.

decimation
A number, N, meaning every Nth sample is acquired in a scope window.

triggermode
Trigger mode for a scope. Values are one of the following:

0	FreeRun (default)
1	Software
2	Signal
3	Scope

numprepostsamples
If this value is less than 0, this is the number of samples to be saved before a trigger event. If this value is greater than 0, this is the number of samples to skip after the trigger event before data acquisition begins.

triggersignal
If *triggermode* is 2 (Signal), identifies the block output signal to use for triggering the scope. Identify the signal with a signal index.

triggerscope
If *triggermode* is 3 (Scope), identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered.

triggerscopesample
If *triggermode* is 3 (Scope), specifies the number of samples to be acquired by the triggering scope before triggering a second scope. This must be a nonnegative value.

triggerlevel
If *triggermode* is 2 (Signal), indicates the value the signal has to cross to trigger the scope to start acquiring data. The trigger level can be crossed with either a rising or falling signal.

triggerslope
If *triggermode* is 2 (Signal), indicates whether the trigger is on a rising or falling signal. Values are:

0	Either rising or falling (default)
1	Rising

2	Falling

Description

The `scopedata` structure holds the data about a scope used in the functions `xPCGetScope` and `xPCSetScope`. In the structure, the fields are as in the various `xPCGetSc*` functions (for example, *`state`* is as in `xPCScGetState`, *`signals`* is as in `xPCScGetSignals`, etc.). The signal vector is an array of the signal identifiers, terminated by -1.

See Also

API functions `xPCSetScope`, `xPCGetScope`, `xPCScGetType`, `xPCScGetState`, `xPCScGetSignals`, `xPCScGetNumSamples`, `xPCScGetDecimation`, `xPCScGetTriggerMode`, `xPCScGetNumPrePostSamples`, `xPCScGetTriggerSignal`, `xPCScGetTriggerScope`, `xPCScGetTriggerLevel`, `xPCScGetTriggerSlope`

xPCAddScope

Create new scope

Prototype

```
void xPCAddScope(int port, int scType, int scNum);
```

Arguments

port	Enter the value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.
scType	Enter the type of scope.
scNum	Enter a number for a new scope. Values are 1, 2, 3. . .

Description

The xPCAddScope function creates a new scope on the target computer. For *scType*, scopes can be of type host or target, depending on the value of *scType*:

- SCTYPE_HOST for type host
- SCTYPE_TARGET for type target
- SCTYPE_FILE for type file

Constants for *scType* are defined in the header file xpcapiconst.h as SCTYPE_HOST, SCTYPE_TARGET, and SCTYPE_FILE.

Calling the xPCAddScope function with *scNum* having the number of an existing scope produces an error. Use xPCGetScopes to find the numbers of existing scopes.

See Also

API functions xPCScAddSignal, xPCScRemSignal, xPCRemScope, xPCSetScope, xPCGetScope, xPCGetScopes

Target object method `SimulinkRealTime.target.addscope`

xPCAverageTET

Return average task execution time

Prototype

```
double xPCAverageTET(int port);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

Return

The `xPCAverageTET` function returns the average task execution time (TET) for the real-time application.

Description

The `xPCAverageTET` function returns the TET for the real-time application. You can use this function when the real-time application is running or when it is stopped.

See Also

API functions `xPCMaximumTET`, `xPCMinimumTET`

Property `AvgTET` of `SimulinkRealTime.target`

xPCCloseConnection

Close RS-232 or TCP/IP communication connection

Prototype

```
void xPCCloseConnection(int port);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

Description

The `xPCCloseConnection` function closes the RS-232 or TCP/IP communication channel opened by `xPCOpenSerialPort`, `xPCOpenTcpIpPort`, or `xPCOpenConnection`. Unlike `xPCClosePort`, it preserves the connection information such that a subsequent call to `xPCOpenConnection` succeeds without the need to resupply communication data such as the IP address or port number. To completely close the communication channel, call `xPCDeRegisterTarget`. Calling the `xPCCloseConnection` function followed by calling `xPCDeRegisterTarget` is equivalent to calling `xPCClosePort`.

Note: RS-232 communication type will be removed in a future release. Use TCP/IP instead.

See Also

API functions `xPCOpenConnection`, `xPCOpenSerialPort`, `xPCOpenTcpIpPort`, `xPCReOpenPort`, `xPCRegisterTarget`, `xPCDeRegisterTarget`

xPCClosePort

Close RS-232 or TCP/IP communication connection

Prototype

```
void xPCClosePort(int port);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

Description

The `xPCClosePort` function closes the RS-232 or TCP/IP communication channel opened by either `xPCOpenSerialPort` or by `xPCOpenTcpIpPort`. Calling this function is equivalent to calling `xPCCloseConnection` and `xPCDeRegisterTarget`.

Note: RS-232 communication type will be removed in a future release. Use TCP/IP instead.

See Also

API functions `xPCOpenSerialPort`, `xPCOpenTcpIpPort`, `xPCReOpenPort`, `xPCOpenConnection`, `xPCCloseConnection`, `xPCRegisterTarget`, `xPCDeRegisterTarget`

Target object method `SimulinkRealTime.target.close`

xPCDeRegisterTarget

Delete target communication properties from Simulink Real-Time API library

Prototype

```
void xPCDeRegisterTarget(int port);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

Description

The `xPCDeRegisterTarget` function causes the Simulink Real-Time API library to completely "forget" about the target communication properties. You use this at the end of a session in which you use `xPCOpenConnection` and `xPCCloseConnection` to connect and disconnect from the target without entering the properties each time. It works similarly to `xPCClosePort`, but does not close the connection to the target computer. Before calling this function, you must first call the function `xPCCloseConnection` to close the connection to the target computer. The combination of calling the `xPCCloseConnection` and `xPCDeRegisterTarget` functions has the same result as calling `xPCClosePort`.

See Also

API functions `xPCRegisterTarget`, `xPCOpenTcpIpPort`, `xPCOpenSerialPort`, `xPCClosePort`, `xPCReOpenPort`, `xPCOpenConnection`, `xPCCloseConnection`, `xPCTargetPing`

xPCErrorMsg

Return text description for error message

Prototype

```
char *xPCErrorMsg(int error_number, char *error_message);
```

Arguments

error_number	Enter the constant of an error.
error_message	The `xPCErrorMsg` function copies the error message string into the buffer pointed to by *error_message*. *error_message* is then returned. You can later use *error_message* in a function such as `printf`. If *error_message* is NULL, the `xPCErrorMsg` function returns a pointer to a statically allocated string.

Return

The `xPCErrorMsg` function returns a string associated with the error *error_number*.

Description

The `xPCErrorMsg` function returns *error_message*, which makes it convenient to use in a `printf` or similar statement. Use the `xPCGetLastError` function to get the constant for which you are getting the message.

See Also

API functions `xPCSetLastError`, `xPCGetLastError`

xPCFreeAPI

Unload Simulink Real-Time DLL

Prototype

```
void xPCFreeAPI(void);
```

Description

The `xPCFreeAPI` function unloads the Simulink Real-Time dynamic link library. You must execute this function once at the end of the application to unload the Simulink Real-Time API DLL. This frees the memory allocated to the functions. This function is defined in the file `xpcinitfree.c`. Link this file with your application.

See Also

API functions `xPCInitAPI`, `xPCNumLogWraps`, `xPCNumLogSamples`, `xPCMaxLogSamples`, `xPCGetStateLog`, `xPCGetTETLog`, `xPCSetLogMode`, `xPCGetLogMode`

xPCFSCD

Change current folder on target computer to specified path

Prototype

```
void xPCFSCD(int port, char *dir);
```

Arguments

port	Enter the value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.
dir	Enter the path on the target computer to change to.

Description

The xPCFSCD function changes the current folder on the target computer to the path specified in *dir*. Use the xPCFSGetPWD function to show the current folder of the target computer.

See Also

API function xPCFSGetPWD

File object method SimulinkRealTime.fileSystem.cd

xPCFSCloseFile

Close file on target computer

Prototype

```
void xPCFSCloseFile(int port, int fileHandle);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
fileHandle	Enter the file handle of an open file on the target computer.

Description

The `xPCFSCloseFile` function closes the file associated with *`fileHandle`* on the target computer. *`fileHandle`* is the handle of a file previously opened by the `xPCFSOpenFile` function.

See Also

API functions `xPCFSOpenFile`, `xPCFSReadFile`, `xPCFSWriteFile`

File object method `SimulinkRealTime.fileSystem.fclose`

xPCFSDir

Get contents of specified folder on target computer

Prototype

```
void xPCFSDir(int port, const char *path, char *data, int numbytes);
```

Arguments

port Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

path Enter the path on the target computer.

data The contents of the folder are stored in *data*, whose allocated size is specified in *numbytes*.

numbytes Enter the size, in bytes, of the array *data*.

Description

The `xPCFSDir` function copies the contents of the target computer folder specified by *path* into `data`. The `xPCFSDir` function returns the listing in the *data* array, which must be of size *numbytes*. Use the `xPCFSDirSize` function to obtain the size of the folder listing for the *numbytes* parameter.

See Also

API function `xPCFSDirSize`

File object method `SimulinkRealTime.fileSystem.dir`

xPCFSDirItems

Get contents of specified folder on target computer

Prototype

```
void xPCFSDirItems(int port, const char *path, dirStruct *dirs, int numDirItems);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
path	Enter the path on the target computer.
dirs	Enter the structure to contain the contents of the folder.
numDirItems	Enter the number of items in the folder.

Description

The `xPCFSDirItems` function copies the contents of the target computer folder specified by *path*. The `xPCFSDirItems` function copies the listing into the *dirs* structure, which must be of size *numDirItems*. Use the `xPCFSDirStructSize` function to obtain the size of the folder for the *numDirItems* parameter.

See Also

API functions `xPCFSDirStructSize`, `dirStruct`

File object method `SimulinkRealTime.fileSystem.dir`

xPCFSDirSize

Return size of specified folder listing on target computer

Prototype

```
int xPCFSDirSize(int port, const char *path);
```

Arguments

port	Enter the value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.
path	Enter the folder path on the target computer.

Return

The xPCFSDirSize function returns the size, in bytes, of the specified folder listing. If this function detects an error, it returns -1.

Description

The xPCFSDirSize function returns the size, in bytes, of the buffer required to list the folder contents on the target computer. Use this size as the *numbytes* parameter in the xPCFSDir function.

See Also

API function xPCFSDirItems

File object method SimulinkRealTime.fileSystem.dir

xPCFSDirStructSize

Get number of items in folder

Prototype

```
int xPCFSDirStructSize(int port, const char *path);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
path	Enter the folder path on the target computer.

Return

The `xPCFSDirStructSize` function returns the number of items in the folder on the target computer. If this function detects an error, it returns `-1`.

Description

The `xPCFSDirStructSize` function returns the number of items in the folder on the target computer. Use this size as the *`numDirItems`* parameter in the `xPCFSDirItems` function.

See Also

API function `xPCFSDir`

File object method `SimulinkRealTime.fileSystem.dir`

xPCFSDiskInfo

Information about target computer file system

Prototype

```
diskinfo xPCFSDiskInfo(int  port, const char *driveletter);
```

Arguments

port — Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

driveletter — Enter the drive letter of the file system for which you want information.

Description

The `xPCFSDiskInfo` function returns disk information for the file system of the specified target computer drive, *driveletter*. This function returns this information in the `diskinfo` structure.

See Also

API structure `SimulinkRealTime.fileSystem.diskinfo`

xPCFSFileInfo

Return information for open file on target computer

Prototype

```
fileinfo xPCFSFileInfo(int port, int fileHandle);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
fileHandle	Enter the file handle of an open file on the target computer.

Description

The `xPCFSFileInfo` function returns information about the specified open file, `filehandle`, in a structure of type `fileinfo`.

See Also

Structure `SimulinkRealTime.fileSystem.fileinfo`

xPCFSGetError

Get text description for error number on target computer file system

Prototype

```
void xPCFSGetError(int port, unsigned int error_number,
char *error_message);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
error_number	Enter the constant of an error.
error_message	The string of the message associated with the error *error_number* is stored in *error_message*.

Description

The `xPCFSGetError` function gets the *error_message* associated with *error_number*. This enables you to use the error message in a `printf` or similar statement.

xPCFSGetFileSize

Return size of file on target computer

Prototype

```
int xPCFSGetFileSize(int port, int fileHandle);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
fileHandle	Enter the file handle of an open file on the target computer.

Return

Return the size of the specified file in bytes. If this function detects an error, it returns `-1`.

Description

The `xPCFSGetFileSize` function returns the size, in bytes, of the file associated with *fileHandle* on the target computer. *fileHandle* is the handle of a file previously opened by the `xPCFSOpenFile` function.

See Also

API functions `xPCFSOpenFile`, `xPCFSReadFile`

File object methods `SimulinkRealTime.fileSystem.fopen` and `SimulinkRealTime.fileSystem.fread`

xPCFSGetPWD

Get current folder of target computer

Prototype

```
void xPCFSGetPWD(int port, char *pwd);
```

Arguments

`port` Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

`pwd` The path of the current folder is stored in *`pwd`*.

Description

The `xPCFSGetPWD` function places the path of the current folder on the target computer in *`pwd`*, which must be allocated by the caller.

See Also

File object method `SimulinkRealTime.fileSystem.pwd`

xPCFSMKDIR

Create new folder on target computer

Prototype

```
void xPCFSMKDIR(int port, const char *dirname);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
dirname	Enter the name of the folder to create on the target computer.

Description

The `xPCFSMKDIR` function creates the folder *`dirname`* in the current folder of the target computer.

See Also

API function `xPCFSGetPWD`

File object method `SimulinkRealTime.fileSystem.mkdir`

xPCFSOpenFile

Open file on target computer

Prototype

```
int xPCFSOpenFile(int port, const char *filename,
const char *permission);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
filename	Enter the name of the file to open on the target computer.
permission	Enter the read/write permission with which to open the file. Values are `r` (read) or `w` (read/write).

Return

The `xPCFSOpenFile` function returns the file handle for the opened file. If function detects an error, it returns `-1`.

Description

The `xPCFSOpenFile` function opens the specified file, *`filename`*, on the target computer. If the file does not exist, the `xPCFSOpenFile` function creates *`filename`*, then opens it. You can open a file for read or read/write access.

See Also

API functions `xPCFSCloseFile`, `xPCFSGetFileSize`, `xPCFSReadFile`, `xPCFSWriteFile`

File object methods `SimulinkRealTime.fileSystem.fclose`, `SimulinkRealTime.fileSystem.filetable`, `SimulinkRealTime.fileSystem.fwrite` `SimulinkRealTime.fileSystem.fopen` and `SimulinkRealTime.fileSystem.fread`

xPCFSReadFile

Read open file on target computer

Prototype

```
void xPCFSReadFile(int port, int fileHandle, int start,
int numbytes, unsigned char *data);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
fileHandle	Enter the file handle of an open file on the target computer.
start	Enter an offset from the beginning of the file from which this function can start to read.
numbytes	Enter the number of bytes this function is to read from the file.
data	The contents of the file are stored in *data*.

Description

The `xPCFSReadFile` function reads an open file on the target computer and places the results of the read operation in the array *data*. *fileHandle* is the file handle of a file previously opened by `xPCFSOpenFile`. You can specify that the read operation begin at the beginning of the file (default) or at a certain offset into the file (*start*). The *numbytes* parameter specifies how many bytes the `xPCFSReadFile` function is to read from the file.

See Also

API functions `xPCFSCloseFile`, `xPCFSGetFileSize`, `xPCFSOpenFile`, `xPCFSWriteFile`

File object methods `SimulinkRealTime.fileSystem.fopen` and `SimulinkRealTime.fileSystem.fread`

xPCFSRemoveFile

Remove file from target computer

Prototype

```
void xPCFSRemoveFile(int port, const char *filename);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
filename	Enter the name of a file on the target computer.

Description

The `xPCFSRemoveFile` function removes the file named *`filename`* from the target computer file system. *`filename`* can be a relative or absolute path name on the target computer.

See Also

File object method `SimulinkRealTime.fileSystem.removefile`

xPCFSRMDIR

Remove folder from target computer

Prototype

```
void xPCFSRMDIR(int port, const char *dirname);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
dirname	Enter the name of a folder on the target computer.

Description

The `xPCFSRMDIR` function removes a folder named ***dirname*** from the target computer file system. ***dirname*** can be a relative or absolute path-name on the target computer.

See Also

File object method `SimulinkRealTime.fileSystem.rmdir`

xPCFSScGetFilename

Get name of file for scope

Prototype

```
const char *xPCFSScGetFilename(int port, int scNum, char *filename);
```

Arguments

port Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

scNum Enter the scope number.

filename The name of the file for the specified scope is stored in *filename*.

Return

Returns the value of *filename*, the name of the file for the scope.

Description

The `xPCFSScGetFilename` function returns the name of the file to which scope *scNum* will save signal data. *filename* points to a caller-allocated character array to which the filename is copied.

See Also

API function `xPCFSScSetFilename`

Property `Filename` of `SimulinkRealTime.fileSystem`

xPCFSScGetWriteMode

Get write mode of file for scope

Prototype

```
int xPCFSScGetWriteMode(int port, int scNum);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.

Return

Returns the number indicating the write mode. Values are

0	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 have the actual file size (the file contents, however, will be intact).
1	Commit mode. Each file write operation simultaneously updates the FAT entry for the file. This mode is slower, but the file system maintains the actual file size.

Description

The `xPCFSScGetWriteMode` function returns the write mode of the file for the scope.

See Also

API function `xPCFSScSetWriteMode`

Property `WriteMode` of `SimulinkRealTime.fileSystem`

xPCFSScGetWriteSize

Get block write size of data chunks

Prototype

```
unsigned int xPCFSScGetWriteSize(int port, int scNum);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.

Return

Returns the block size, in bytes, of the data chunks.

Description

The `xPCFSScGetWriteSize` function gets the block size, in bytes, of the data chunks.

See Also

API function `xPCFSScSetWriteSize`

Property `WriteSize` of `SimulinkRealTime.fileSystem`

xPCFSScSetFilename

Specify name for file to contain signal data

Prototype

```
void xPCFSScSetFilename(int port, int scNum,
const char *filename);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.
filename	Enter the name of a file to contain the signal data.

Description

The `xPCFSScSetFilename` function sets the name of the file to which the scope will save the signal data. The Simulink Real-Time software creates this file in the target computer file system. Note that you can only call this function when the scope is stopped.

See Also

API function `xPCFSScGetFilename`

Property `Filename` of `SimulinkRealTime.fileSystem`

xPCFSScSetWriteMode

Specify when file allocation table entry is updated

Prototype

```
void xPCFSScSetWriteMode(int port, int scNum, int writeMode);
```

Arguments

port Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

scNum Enter the scope number.

writeMode Enter an integer for the write mode:

0	Enables lazy write mode
1	Enables commit write mode

Description

The `xPCFSScSetWriteMode` function specifies when a file allocation table (FAT) entry is updated. Both modes write the signal data to the file, as follows:

0 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 have the actual file size (the file contents, however, will be intact).

1 Commit mode. Each file write operation simultaneously updates the FAT entry for the file. This mode is slower, but the file system maintains the actual file size.

See Also

API function `xPCFSScGetWriteMode`

Property `WriteMode` of `SimulinkRealTime.fileSystem`

xPCFSScSetWriteSize

Specify that memory buffer collect data in multiples of write size

Prototype

```
void xPCFSScSetWriteSize(int port, int scNum, unsigned int
writeSize);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.
writeSize	Enter the block size, in bytes, of the data chunks.

Description

The `xPCFSScSetWriteSize` function specifies that a memory buffer collect data in multiples of *`writeSize`*. By default, this parameter is 512 bytes, which is the typical disk sector size. Using a block size that is the same as the disk sector size provides better performance. *`writeSize`* must be a multiple of `512`.

See Also

API function `xPCFSScGetWriteSize`

Property `WriteSize` of `SimulinkRealTime.fileSystem`

xPCFSWriteFile

Write to file on target computer

Prototype

```
void xPCFSWriteFile(int port, int fileHandle, int numbytes,
const unsigned char *data);
```

Arguments

port	Enter the value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.
fileHandle	Enter the file handle of an open file on the target computer.
numbytes	Enter the number of bytes this function is to write into the file.
data	The contents to write to *fileHandle* are stored in *data*.

Description

The xPCFSWriteFile function writes the contents of the array *data* to the file specified by *fileHandle* on the target computer. The *fileHandle* parameter is the handle of a file previously opened by xPCFSOpenFile. *numbytes* is the number of bytes to write to the file.

See Also

API functions xPCFSCloseFile, xPCFSGetFileSize, xPCFSOpenFile, xPCFSReadFile

xPCGetAPIVersion

Get version number of Simulink Real-Time API

Prototype

```
const char *xPCGetAPIVersion(void);
```

Return

The `xPCGetApiVersion` function returns a string with the version number of the Simulink Real-Time kernel on the target computer.

Description

The `xPCGetApiVersion` function returns a string with the version number of the Simulink Real-Time kernel on the target computer. The string is a constant string within the API DLL. Do not modify this string.

See Also

API function `xPCGetTargetVersion`

xPCGetAppName

Return real-time application name

Prototype

```
char *xPCGetAppName(int port, char *model_name);
```

Arguments

port Enter the value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.

model_name The xPCGetAppName function copies the real-time application name string into the buffer pointed to by *model_name*. *model_name* is then returned. You can later use *model_name* in a function such as printf.

Note that the maximum size of the buffer is 256 bytes. To reserve enough space for the application name string, allocate a buffer of size 256 bytes.

Return

The xPCGetAppName function returns a string with the name of the real-time application.

Description

The xPCGetAppName function returns the name of the real-time application. You can use the return value, *model_name*, in a printf or similar statement. In case of error, the name string is unchanged.

Examples

Allocate 256 bytes for the buffer appname.

```
char *appname=malloc(256);
xPCGetAppName(iport,appname);
appname=realloc(appname,strlen(appname)+1);
...
free(appname);
```

See Also

API function `xPCIsAppRunning`

Target object property `Application`

xPCGetEcho

Return display mode for target message window

Prototype

```
int xPCGetEcho(int port);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

Return

The `xPCGetEcho` function returns the number indicating the display mode. Values are

1	Display is on. Messages are displayed in the message display window on the target.
0	Display is off.

Return

The `xPCGetEcho` function the display mode of the target computer using communication channel *port*. If the function detects an error, it returns `-1`.

Description

The `xPCGetEcho` function returns the display mode of the target computer using communication channel *port*. Messages include the status of downloading the real-time application, changes to parameters, and changes to scope signals.

See Also

API function `xPCSetEcho`

xPCGetExecTime

Return real-time application execution time

Prototype

```
double xPCGetExecTime(int port);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

Return

The `xPCGetExecTime` function returns the current execution time for a real-time application. If the function detects an error, it returns `-1`.

Description

The `xPCGetExecTime` function returns the current execution time for the running real-time application. If the real-time application is stopped, the value is the last running time when the real-time application was stopped. If the real-time application is running, the value is the current running time.

See Also

API functions `xPCSetStopTime`, `xPCGetStopTime`

Property `ExecTime` of `SimulinkRealTime.target`

xPCGetLastError

Return constant of last error

Prototype

```
int xPCGetLastError(void);
```

Return

The `xPCGetLastError` function returns the error constant for the last reported error. If the function did not detect an error, it returns 0.

Description

The `xPCGetLastError` function returns the constant of the last reported error by another API function. This value is reset every time you call a new function. Therefore, you should check this constant value immediately after a call to an API function. For a list of error constants and messages, see “C API Error Messages”.

See Also

API functions `xPCErrorMsg`, `xPCSetLastError`

xPCGetLoadTimeOut

Return timeout value for communication between development and target computers

Prototype

```
int xPCGetLoadTimeOut(int port);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

Return

The `xPCGetLoadTimeOut` function returns the number of seconds allowed for the communication between the development computer and real-time application. If the function detects an error, it returns `-1`.

Description

The `xPCGetLoadTimeOut` function returns the number of seconds allowed for the communication between the development computer and the real-time application. When a Simulink Real-Time API function initiates communication between the development and target computers, it waits for a certain amount of time before checking to see if the communication is complete. In the case where communication with the target computer is not complete, the function returns a timeout error.

For example, when you load a new real-time application onto the target computer, the function `xPCLoadApp` waits for a certain amount of time before checking to see if the initialization of the real-time application is complete. In the case where initialization of the real-time application is not complete, the function `xPCLoadApp` returns a timeout error. By default, `xPCLoadApp` checks for the readiness of the target computer for up to 5 seconds. However, for larger models or models requiring longer initialization (for example, those with thermocouple boards), the default might not be long enough and a spurious timeout is generated. Other functions that communicate with the target computer will wait for *`timeOut`* seconds before declaring a timeout event. The function `xPCSetLoadTimeOut` sets the timeout to a different number.

Use the `xPCGetLoadTimeOut` function if you suspect that the current number of seconds (the timeout value) is too short. Then use the `xPCSetLoadTimeOut` function to set the timeout to a higher number.

See Also

API functions `xPCLoadApp`, `xPCSetLoadTimeOut`

`xPCUnloadApp`

"Increase the Time for Downloads"

xPCGetLogMode

Return logging mode and increment value for real-time application

Prototype

```
lgmode xPCGetLogMode(int port);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

Return

The `xPCGetLogMode` function returns the logging mode in the `lgmode` structure. If the logging mode is `1` (`LGMOD_VALUE`), this function also returns an increment value in the `lgmode` structure. If an error occurs, this function returns `-1`.

Description

The `xPCGetLogMode` function gets the logging mode and increment value for the current real-time application. The increment (difference in amplitude) value is measured between logged data points. A data point is logged only when an output signal or a state changes by the increment value.

See Also

API function `xPCSetLogMode`

API structure `lgmode`

xPCGetNumOutputs

Return number of outputs

Prototype

```
int xPCGetNumOutputs(int port);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

Return

The `xPCGetNumOutputs` function returns the number of outputs in the current real-time application. If the function detects an error, it returns `-1`.

Description

The `xPCGetNumOutputs` function returns the number of outputs in the real-time application. The number of outputs equals the sum of the input signal widths of the output blocks at the root level of the Simulink model.

See Also

API functions `xPCGetOutputLog`, `xPCGetNumStates`, `xPCGetStateLog`

xPCGetNumParams

Return number of tunable parameters

Prototype

```
int xPCGetNumParams(int port);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

Return

The `xPCGetNumParams` function returns the number of tunable parameters in the real-time application. If the function detects an error, it returns `-1`.

Description

The `xPCGetNumParams` function returns the number of tunable parameters in the real-time application. Use this function to see how many parameters you can get or modify.

See Also

API functions `xPCGetParamIdx`, `xPCSetParam`, `xPCGetParam`, `xPCGetParamName`, `xPCGetParamDims`

Property `NumParameters` of `SimulinkRealTime.target`

xPCGetNumScopes

Return number of scopes added to real-time application

Prototype

```
int  xPCGetNumScopes(int port);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

Return

The `xPCGetNumScopes` function returns the number of scopes that have been added to the real-time application. If the function detects an error, it returns `-1`.

Description

The `xPCGetNumScopes` function returns the number of scopes that have been added to the real-time application.

xPCGetNumScSignals

Returns number of signals added to specific scope

Prototype

```
int xPCGetNumScSignals(int port, int scopeId);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scopeId	Enter the ID number of the scope for which you want to get the number of added signals.

Return

The `xPCGetNumScSignals` function returns the number of signals that have been added to the scope, *`scopeID`*. If the function detects an error, it returns `-1`.

Description

The `xPCGetNumScSignals` function returns the number of signals that have been added to the scope, *`scopeID`*.

xPCGetNumSignals

Return number of signals

Prototype

```
int xPCGetNumSignals(int port);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

Return

The `xPCGetNumSignals` function returns the number of signals in the real-time application. If the function detects an error, it returns `-1`.

Description

The `xPCGetNumSignals` function returns the total number of signals in the real-time application that can be monitored from the development computer. Use this function to see how many signals you can monitor.

See Also

API functions `xPCGetSignalIdx`, `xPCGetSignal`, `xPCGetSignals`, `xPCGetSignalName`, `xPCGetSignalWidth`

Property `NumSignals` of `SimulinkRealTime.target`

xPCGetNumStates

Return number of states

Prototype

```
int xPCGetNumStates(int port);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

Return

The `xPCGetNumStates` function returns the number of states in the real-time application. If the function detects an error, it returns `-1`.

Description

The `xPCGetNumStates` function returns the number of states in the real-time application.

See Also

API functions `xPCGetStateLog`, `xPCGetNumOutputs`, `xPCGetOutputLog`

Property `StateLog` of `SimulinkRealTime.target`

xPCGetOutputLog

Copy output log data to array

Prototype

```
void xPCGetOutputLog(int port, int first_sample, int num_samples,
int decimation, int output_id, double *output_data);
```

Arguments

port Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

first_sample Enter the index of the first sample to copy.

num_samples Enter the number of samples to copy from the output log.

decimation Select whether to copy every sample value or every `N`th value.

output_id Enter an output identification number.

output_data The log is stored in *output_data*, whose allocation is the responsibility of the caller.

Description

The `xPCGetOutputLog` function gets the output log and copies that log to an array. You get the data for each output signal in turn by specifying *output_id*. Output IDs range from `0` to (`N`-1), where `N` is the return value of `xPCGetNumOutputs`. Entering `1` for *decimation* copies all values. Entering `N` copies every `N`th value.

For *first_sample*, the sample indices range from `0` to (`N`-1), where `N` is the return value of `xPCNumLogSamples`. Get the maximum number of samples by calling the function `xPCNumLogSamples`.

Note that the real-time application must be stopped before you get the number.

See Also

API functions xPCNumLogWraps, xPCNumLogSamples, xPCMaxLogSamples, xPCGetNumOutputs, xPCGetStateLog, xPCGetTETLog, xPCGetTimeLog

Target object method SimulinkRealTime.target.getlog

Property OutputLog of SimulinkRealTime.target

xPCGetParam

Get parameter value and copy it to array

Prototype

```
void xPCGetParam(int port, int paramIndex, double *paramValue);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
paramIndex	Enter the index for a parameter.
paramValue	The function returns a parameter value as an array of doubles.

Description

The `xPCGetParam` function returns the parameter as an array in *paramValue*. *paramValue* must be large enough to hold the parameter. You can query the size by calling the function `xPCGetParamDims`. Get the parameter index by calling the function `xPCGetParamIdx`. The parameter matrix is returned as a vector, with the conversion being done in column-major format. It is also returned as a double, regardless of the data type of the actual parameter.

For *paramIndex*, values range from `0` to (`N`-1), where `N` is the return value of `xPCGetNumParams`.

See Also

API functions `xPCSetParam`, `xPCGetParamDims`, `xPCGetParamIdx`, `xPCGetNumParams`

`SimulinkRealTime.target.getparamid`

Properties `ShowParameters` and `Parameters` of `SimulinkRealTime.target`

xPCGetParamDims

Get row and column dimensions of parameter

Prototype

```
void xPCGetParamDims(int port, int paramIndex, int *dimension);
```

Arguments

port Enter the value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.

paramIndex Parameter index.

dimension Dimensions (row, column) of a parameter.

Description

The xPCGetParamDims function gets the dimensions (row, column) of a parameter with *paramIndex* and stores them in *dimension*, which must have at least two elements.

For *paramIndex*, values range from 0 to (N-1), where N is the return value of xPCGetNumParams.

See Also

API functions xPCGetParamIdx, xPCGetParamName, xPCSetParam, xPCGetParam, xPCGetNumParams

SimulinkRealTime.target.getparamid

Properties ShowParameters and Parameters of SimulinkRealTime.target

xPCGetParamIdx

Return parameter index

Prototype

```
int xPCGetParamIdx(int port, const char *blockName,
const char *paramName);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
blockName	Enter the full block path generated by Simulink Coder™.
paramName	Enter the parameter name for a parameter associated with the block.

Return

The `xPCGetParamIdx` function returns the parameter index for the parameter name. If the function detects an error, it returns `-1`.

Description

The `xPCGetParamIdx` function returns the parameter index for the parameter name (*paramName)* associated with a Simulink block (*blockName*). Both *blockName* and *paramName* must be identical to those generated at real-time application building time. The block names should be referenced from the file `model_namept.m` in the generated code, where *model_name* is the name of the model. Note that a block can have one or more parameters.

See Also

API functions `xPCGetParamDims`, `xPCGetParamName`, `xPCGetParam`

`SimulinkRealTime.target.getparamid`

Properties `ShowParameters` and `Parameters` of `SimulinkRealTime.target`

xPCGetParamName

Get name of parameter

Prototype

```
void xPCGetParamName(int port, int paramIdx, char *blockName, char
*paramName);
```

Arguments

port Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

paramIdx Enter a parameter index.

blockName String with the full block path generated by Simulink Coder.

paramName Name of a parameter for a specific block.

Description

The `xPCGetParamName` function gets the parameter name and block name for a parameter with the index *paramIdx*. The block path and name are returned and stored in *blockName*, and the parameter name is returned and stored in *paramName*. You must allocate enough space for both *blockName* and *paramName*. If the *paramIdx* is invalid, `xPCGetLastError` returns nonzero, and the strings are unchanged. Get the parameter index from the function `xPCGetParamIdx`.

See Also

API functions `xPCGetParam`, `xPCGetParamDims`, `xPCGetParamIdx`

Properties `ShowParameters` and `Parameters` of `SimulinkRealTime.target`

xPCGetSampleTime

Return real-time application sample time

Prototype

```
double xPCGetSampleTime(int port);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

Return

The `xPCGetSampleTime` function returns the sample time, in seconds, of the real-time application. If the function detects an error, it returns `-1`.

Description

The `xPCGetSampleTime` function returns the sample time, in seconds, of the real-time application. You can get the error by using the function `xPCGetLastError`.

See Also

API function `xPCSetSampleTime`

Property `SampleTime` of `SimulinkRealTime.target`

xPCGetScope

Get and copy scope data to structure

Prototype

```
scopedata xPCGetScope(int port, int scNum);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.

Return

The `xPCGetScope` function returns a structure of type `scopedata`.

Description

Note: The `xPCGetScope` function will be removed in a future release. Use the `xPCScGet`*`ScopePropertyName`* functions to access property values instead. For example, to get the number of samples being acquired in one data acquisition cycle, use `xPCScGetNumSamples`.

The `xPCGetScope` function gets properties of a scope with *`scNum`* and copies the properties into a structure with type `scopedata`. You can use this function in conjunction with `xPCSetScope` to change several properties of a scope at one time. See `scopedata` for a list of properties. Use the `xPCGetScope` function to get the scope number.

See Also

API functions `xPCSetScope`, `scopedata`

Target object method `SimulinkRealTime.target.getscope`

xPCGetScopeList

Get and copy list of scope numbers

Prototype

```
void xPCGetScopeList(int port, int *data);
```

Arguments

port Enter the value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.

data List of scope numbers in an integer array (allocated by the caller) as a list of unsorted integers.

Description

The xPCGetScopeList function gets the list of scopes currently defined. *data* must be large enough to hold the list of scopes. You can query the size by calling the function xPCGetNumScopes.

Note: Use the xPCGetScopeList function instead of the xPCGetScopes function. The xPCGetScopes will be removed in a future release.

xPCGetScopes

Get and copy list of scope numbers

Prototype

```
void xPCGetScopes(int port, int *data);
```

Arguments

port Enter the value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.

data List of scope numbers in an integer array (allocated by the caller) as a list of unsorted integers and terminated by -1.

Description

The xPCGetScopes function gets the list of scopes currently defined. You can use the constant MAX_SCOPES (defined in xpcapiconst.h) as the size of *data*. This is currently set to 30 scopes.

Note: This function will be removed in a future release. Use the xPCGetScopeList function instead.

See Also

API functions xPCSetScope, xPCGetScope, xPCScGetSignals

Property Scopes of SimulinkRealTime.target

xPCGetSessionTime

Return length of time Simulink Real-Time kernel has been running

Prototype

```
double xPCGetSessionTime(int port);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

Return

The `xPCGetSessionTime` function returns the amount of time in seconds that the Simulink Real-Time kernel has been running on the target computer. If the function detects an error, it returns `-1`.

Description

The `xPCGetSessionTime` function returns, as a double, the amount of time in seconds that the Simulink Real-Time kernel has been running. This value is also the time that has elapsed since you last booted the target computer.

xPCGetSignal

Return value of signal

Prototype

```
double xPCGetSignal(int port, int sigNum);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
sigNum	Enter a signal number.

Return

The `xPCGetSignal` function returns the current value of signal *`sigNum`*. If the function detects an error, it returns `-1`.

Description

The `xPCGetSignal` function returns the current value of a signal. For vector signals, use `xPCGetSignals` rather than call this function multiple times. Use the `xPCGetSignalIdx` function to get the signal number.

See Also

API function `xPCGetSignals`

Property `Signals` of `SimulinkRealTime.target`

xPCGetSignalIdx

Return index for signal

Prototype

```
int xPCGetSignalIdx(int port, const char *sigName);
```

Arguments

port	Enter the value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.
sigName	Enter a signal name.

Return

The xPCGetSignalIdx function returns the index for the signal with name *sigName*. If the function detects an error, it returns -1.

Description

The xPCGetSignalIdx function returns the index of a signal. The name must be identical to the name generated when the application was built. You should reference the name from the file model_namebio.m in the generated code, where *model_name* is the name of the model. The creator of the application should already know the signal name.

See Also

API functions xPCGetSignalName, xPCGetSignalWidth, xPCGetSignal, xPCGetSignals

Target object method SimulinkRealTime.target.getsignalid

xPCGetSigIdxfromLabel

Return array of signal indices

Prototype

```
int xPCGetSigIdxfromLabel(int port, const char *sigLabel, int *sigIds);
```

Arguments

port Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

sigLabel String with the name of a signal label.

sigIds Return array of signal indices.

Return

If `xPCGetSigIdxfromLabel` finds a signal, it fills an array *sigIds* with signal indices and returns `0`. If it finds no signal, it returns `-1`.

Description

The `xPCGetSigIdxfromLabel` function returns in *sigIds* the array of signal indices for signal *sigName*. This function assumes that you have labeled the signal for which you request the indices (see the **Signal name** parameter of the "Signal Properties Controls"). Note that the Simulink Real-Time software refers to Simulink signal names as signal labels. The creator of the application should already know the signal name/label. Signal labels must be unique.

sigIds must be large enough to contain the array of indices. You can use the `xPCGetSigLabelWidth` function to get the required amount of memory to be allocated by the `sigIds` array.

See Also

API functions `xPCGetSignalLabel`, `xPCGetSigLabelWidth`

xPCGetSignalLabel

Copy label of signal to character array

Prototype

```
char * xPCGetSignalLabel(int port, int sigIdx, char *sigLabel);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
sigIdx	Enter signal index.
sigLabel	Return signal label associated with signal index, *sigIdx*.

Return

The `xPCGetSignalLabel` function returns the label of the signal.

Description

The `xPCGetSignalLabel` function copies and returns the signal label, including the block path, of a signal with *sigIdx*. The result is stored in *sigLabel*. If *sigIdx* is invalid, `xPCGetLastError` returns a nonzero value, and *sigLabel* is unchanged. The function returns *sigLabel*, which makes it convenient to use in a `printf` or similar statement. This function assumes that you already know the signal index. Signal labels must be unique.

This function assumes that you have labeled the signal for which you request the index (see the **Signal name** parameter of the “Signal Properties Controls”). Note that the Simulink Real-Time software refers to Simulink signal names as signal labels. The creator of the application should already know the signal name/label.

See Also

API functions `xPCGetSigIdxfromLabel`, `xPCGetSigLabelWidth`

xPCGetSigLabelWidth

Return number of elements in signal

Prototype

```
int xPCGetSigLabelWidth(int port, const char *sigName);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
sigName	String with the name of a signal.

Return

The `xPCGetSigLabelWidth` function returns the number of elements that the signal `sigName` contains. If the function detects an error, it returns `-1`.

Description

The `xPCGetSigLabelWidth` function returns the number of elements that the signal *sigName* contains. This function assumes that you have labeled the signal for which you request the elements (see the **Signal name** parameter of the “Signal Properties Controls”). Note that the Simulink Real-Time software refers to Simulink signal names as signal labels. The creator of the application should already know the signal name/ label. Signal labels must be unique.

See Also

API functions `xPCGetSigIdxfromLabel`, `xPCGetSignalLabel`

xPCGetSignalName

Copy name of signal to character array

Prototype

```
char *xPCGetSignalName(int port, int sigIdx, char *sigName);
```

Arguments

port	Enter the value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.
sigIdx	Enter a signal index.
sigName	String with the name of a signal.

Return

The xPCGetSignalName function returns the name of the signal.

Description

The xPCGetSignalName function copies and returns the signal name, including the block path, of a signal with *sigIdx*. The result is stored in *sigName*. If *sigIdx* is invalid, xPCGetLastError returns a nonzero value, and *sigName* is unchanged. The function returns *sigName*, which makes it convenient to use in a printf or similar statement. This function assumes that you already know the signal index.

See Also

API functions xPCGetSignalIdx, xPCGetSignalWidth, xPCGetSignal, xPCGetSignals

Properties ShowSignals and Signals of SimulinkRealTime.target

xPCGetSignals

Return vector of signal values

Prototype

```
int xPCGetSignals(int port, int numSignals, const int *signals,
double *values);
```

Arguments

port Enter the value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.

numSignals Enter the number of signals to be acquired (that is, the number of values in *signals*).

signals Enter the list of signal numbers to be acquired.

values Returned values are stored in the double array *values*.

Return

The xPCGetSignals function returns 0 if it completes execution without detecting an error. If the function detects an error, it returns -1.

Description

The xPCGetSignals function is the vector version of the function xPCGetSignal. This function returns the values of a vector of signals (up to 1000) as fast as it can acquire them. The signal values may not be at the same time step (for that, define a scope of type SCTYPE_HOST and use xPCScGetData). xPCGetSignal does the same thing for a single signal, and could be used multiple times to achieve the same result. However, the xPCGetSignals function is faster, and the signal values are more likely to be spaced closely together. The signals are converted to doubles regardless of the actual data type of the signal.

For *signals*, the list you provide should be stored in an integer array. Get the signal numbers with the function xPCGetSignalIdx.

See Also

API function xPCGetSignal, xPCGetSignalIdx

Example

To reference signal vector data rather than scalar values, pass a vector of indices for the signal data. For example:

```
/**********************************************************/

/* Assume a signal of width 10, with the blockpath
* mySubsys/mySignal and the signal index s1.
*/

int i;
int sigId[10];
double sigVal[10]; /* Signal values are stored here */

/* Get the ID of the first signal */
sigId[0] = xPCGetSignalIdx(port, "mySubsys/mySignal/s1");

if (sigId[0] == -1) {
/* Handle error */
}

for (i = 1; i < 10; i++) {
     sigId[i] = sigId[0] + i;
}

xPCGetSignals(port, 10, sigId, sigVal);
/* If no error, sigVal should have the signal values */

/***********************************************************/
```

To repeatedly get the signals, repeat the call to xPCGetSignals. If you do not change sigID, you only need to call xPCGetSignalIdx once.

xPCGetSignalWidth

Return width of signal

Prototype

```
int xPCGetSignalWidth(int port, int sigIdx);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
sigIdx	Enter the index of a signal.

Return

The `xPCGetSignalWidth` function returns the signal width for a signal with *`sigIdx`*. If the function detects an error, it returns `-1`.

Description

The `xPCGetSignalWidth` function returns the number of signals for a specified signal index. Although signals are manipulated as scalars, the width of the signal might be useful to reassemble the components into a vector again. A signal's width is the number of signals in the vector.

See Also

API functions `xPCGetSignalIdx`, `xPCGetSignalName`, `xPCGetSignal`, `xPCGetSignals`

xPCGetStateLog

Copy state log values to array

Prototype

```
void xPCGetStateLog(int port, int first_sample, int num_samples,
int decimation, int state_id, double *state_data);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
first_sample	Enter the index of the first sample to copy.
num_samples	Enter the number of samples to copy from the output log.
decimation	Select whether to copy all the sample values or every `N`th value.
state_id	Enter a state identification number.
state_data	The log is stored in *state_data*, whose allocation is the responsibility of the caller.

Description

The `xPCGetStateLog` function gets the state log. It then copies the log into *state_data*. You get the data for each state signal in turn by specifying the *state_id*. State IDs range from `1` to (`N`-1), where `N` is the return value of `xPCGetNumStates`. Entering `1` for *decimation* copies all values. Entering `N` copies every `N`th value. For *first_sample*, the sample indices range from `0` to (`N`-1), where `N` is the return value of `xPCNumLogSamples`. Use the `xPCNumLogSamples` function to get the maximum number of samples.

Note that the real-time application must be stopped before you get the number.

See Also

API functions xPCNumLogWraps, xPCNumLogSamples, xPCMaxLogSamples, xPCGetNumStates, xPCGetOutputLog, xPCGetTETLog, xPCGetTimeLog

SimulinkRealTime.target.getlog

Property StateLog of SimulinkRealTime.target

xPCGetStopTime

Return stop time

Prototype

```
double xPCGetStopTime(int port);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

Return

The `xPCGetStopTime` function returns the stop time as a double, in seconds, of the real-time application. If the function detects an error, it returns `-10.0`. If the stop time is infinity (run forever), this function returns `-1.0`.

Description

The `xPCGetStopTime` function returns the stop time, in seconds, of the real-time application. This is the amount of time the real-time application runs before stopping. If the function detects an error, it returns `-10.0`. You will then need to use the function `xPCGetLastError` to find the error number.

See Also

API function `xPCSetStopTime`

Property `StopTime` of `SimulinkRealTime.target`

xPCGetTargetVersion

Get Simulink Real-Time kernel version

Prototype

```
void xPCGetTargetVersion(int port, char *ver);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
ver	The version is stored in *ver*.

Description

The `xPCGetTargetVersion` function gets a string with the version number of the Simulink Real-Time kernel on the target computer. It then copies that version number into *ver*.

See Also

`xPCGetAPIVersion`

xPCGetTETLog

Copy TET log to array

Prototype

```
void xPCGetTETLog(int port, int first_sample,
int num_samples, int decimation,
double *TET_data);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
first_sample	Enter the index of the first sample to copy.
num_samples	Enter the number of samples to copy from the TET log.
decimation	Select whether to copy all the sample values or every `N`th value.
TET_data	The log is stored in *TET_data*, whose allocation is the responsibility of the caller.

Description

The `xPCGetTETLog` function gets the task execution time (TET) log. It then copies the log into *TET_data*. Entering `1` for *decimation* copies all values. Entering `N` copies every `N`th value. For *first_sample*, the sample indices range from `0` to (`N`-1), where `N` is the return value of `xPCNumLogSamples`. Use the `xPCNumLogSamples` function to get the maximum number of samples.

Note that the real-time application must be stopped before you get the number.

See Also

API functions `xPCNumLogWraps`, `xPCNumLogSamples`, `xPCMaxLogSamples`, `xPCGetNumOutputs`, `xPCGetStateLog`, `xPCGetTimeLog`

`SimulinkRealTime.target.getlog`

Property `TETLog` of `SimulinkRealTime.target`

xPCGetTimeLog

Copy time log to array

Prototype

```
void xPCGetTimeLog(int port, int first_sample, int num_samples,
int decimation, double *time_data);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
first_sample	Enter the index of the first sample to copy.
num_samples	Enter the number of samples to copy from the time log.
decimation	Select whether to copy all the sample values or every `N`th value.
time_data	The log is stored in *time_data*, whose allocation is the responsibility of the caller.

Description

The `xPCGetTimeLog` function gets the time log and copies the log into *time_data*. This is especially relevant in the case of value-equidistant logging, where the logged values might not be uniformly spaced in time. Entering `1` for *decimation* copies all values. Entering `N` copies every `N`th value. For *first_sample*, the sample indices range from `0` to (`N`-1), where `N` is the return value of `xPCNumLogSamples`. Use the `xPCNumLogSamples` function to get the number of samples.

Note that the real-time application must be stopped before you get the number.

See Also

API functions `xPCNumLogWraps`, `xPCNumLogSamples`, `xPCMaxLogSamples`, `xPCGetStateLog`, `xPCGetTETLog`, `xPCSetLogMode`, `xPCGetLogMode`

`SimulinkRealTime.target.getlog`

Property `TimeLog` of `SimulinkRealTime.target`

xPCInitAPI

Initialize Simulink Real-Time DLL

Prototype

```
int xPCInitAPI(void);
```

Return

The `xPCInitAPI` function returns `0` if it completes execution without detecting an error. If the function detects an error, it returns `-1`.

Description

The `xPCInitAPI` function initializes the Simulink Real-Time dynamic link library. You must execute this function once at the beginning of the application to load the Simulink Real-Time API DLL. This function is defined in the file `xpcinitfree.c`. Link this file with your application.

See Also

API functions `xPCFreeAPI`, `xPCNumLogWraps`, `xPCNumLogSamples`, `xPCMaxLogSamples`, `xPCGetStateLog`, `xPCGetTETLog`, `xPCSetLogMode`, `xPCGetLogMode`

xPCIsAppRunning

Return real-time application running status

Prototype

```
int xPCIsAppRunning(int port);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

Return

If the real-time application is stopped, the `xPCIsAppRunning` function returns `0`. If the real-time application is running, this function returns `1`. If the function detects an error, it returns `-1`.

Description

The `xPCIsAppRunning` function returns `1` or `0` depending on whether the real-time application is stopped or running. If the function detects is an error, use the function `xPCGetLastError` to check for the error string constant.

See Also

API function `xPCIsOverloaded`

Property `Status` of `SimulinkRealTime.target`

xPCIsOverloaded

Return target computer overload status

Prototype

```
int xPCIsOverloaded(int port);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

Return

If the real-time application has overloaded the CPU, the `xPCIsOverloaded` function returns `1`. If it has not overloaded the CPU, the function returns `0`. If this function detects error, it returns `-1`.

Description

The `xPCIsOverloaded` function checks if the real-time application has overloaded the target computer and returns `1` if it has and `0` if it has not. If the real-time application is not running, the function returns `0`.

See Also

API function `xPCIsAppRunning`

Property `CPUoverload` of `SimulinkRealTime.target`

xPCIsScFinished

Return data acquisition status for scope

Prototype

```
int xPCIsScFinished(int port, int scNum);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.

Return

If a scope finishes a data acquisition cycle, the `xPCIsScFinished` function returns `1`. If the scope is in the process of acquiring data, this function returns `0`. If the function detects an error, it returns `-1`.

Description

The `xPCIsScFinished` function returns a Boolean value depending on whether scope *scNum* is finished (state of `SCST_FINISHED`) or not. You can also call this function for target scopes; however, because target scopes restart immediately, it is almost impossible to find these scopes in the finished state. Use the `xPCGetScope` function to get the scope number.

See Also

API function `xPCScGetState`

Scope object property `Status`

xPCLoadApp

Load real-time application onto target computer

Prototype

```
void xPCLoadApp(int port, const char *pathstr,
const char *filename);
```

Arguments

port	Enter the value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.
pathstr	Enter the full path to the real-time application file, excluding the file name. For example, in C, use a string like "C:\\work".
filename	Enter the name of a compiled real-time application (*.dlm) without the file extension. For example, in C use a string like "xpcosc".

Description

The xPCLoadApp function loads the compiled real-time application to the target computer. *pathstr* must not contain the trailing backslash. *pathstr* can be set to NULL or to the string 'nopath' if the application is in the current folder. The variable *filename* must not contain the real-time application extension.

Before returning, xPCLoadApp waits for a certain amount of time before checking whether the model initialization is complete. In the case where the model initialization is incomplete, xPCLoadApp returns a timeout error to indicate a connection problem (for example, ETCPREAD). By default, xPCLoadApp checks for target readiness five times, with each attempt taking approximately 1 second (less if the target is ready). However, for larger models or models requiring longer initialization (for example, those with thermocouple boards), the default might not be long enough and a spurious timeout can be generated. The functions xPCGetLoadTimeOut and xPCSetLoadTimeOut control the number of attempts made.

See Also

API functions `xPCStartApp`, `xPCStopApp`, `xPCUnloadApp`, `xPCSetLoadTimeOut`, `xPCGetLoadTimeOut`

Target object method `SimulinkRealTime.target.load`

xPCLoadParamSet

Restore parameter values

Prototype

```
void xPCLoadParamSet(int port, const char *filename);
```

Arguments

port	Enter the value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.
filename	Enter the name of the file that contains the saved parameters.

Description

The xPCLoadParamSet function restores the real-time application parameter values saved in the file *filename*. This file must be located on a local drive of the target computer. The parameter file must have been saved from a previous call to xPCSaveParamSet.

See Also

API function xPCSaveParamSet

xPCMaxLogSamples

Return maximum number of samples that can be in log buffer

Prototype

```
int xPCMaxLogSamples(int port);
```

Arguments

port	Enter the value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.

Return

The xPCMaxLogSamples function returns the total number of samples. If the function detects an error, it returns -1.

Description

The xPCMaxLogSamples function returns the total number of samples that can be returned in the logging buffers.

See Also

API functions xPCNumLogSamples, xPCNumLogWraps, xPCGetStateLog, xPCGetOutputLog, xPCGetTETLog, xPCGetTimeLog

Property MaxLogSamples of SimulinkRealTime.target

xPCMaximumTET

Copy maximum task execution time to array

Prototype

```
void xPCMaximumTET(int port, double *data);
```

Arguments

port	Enter the value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.
data	Array of at least two doubles.

Description

The xPCMaximumTET function gets the maximum task execution time (TET) that was achieved during the previous real-time application run. This function also returns the time at which the maximum TET was achieved. The xPCMaximumTET function then copies these values into the *data* array. The maximum TET value is copied into the first element, and the time at which it was achieved is copied into the second element.

See Also

API functions xPCMinimumTET, xPCAverageTET

Property MaxTET of SimulinkRealTime.target

xPCMinimumTET

Copy minimum task execution time to array

Prototype

```
void xPCMinimumTET(int port, double *data);
```

Arguments

port	Enter the value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.
data	Array of at least two doubles.

Description

The xPCMinimumTET function gets the minimum task execution time (TET) that was achieved during the previous real-time application run. This function also returns the time at which the minimum TET was achieved. The xPCMinimumTET function then copies these values into the *data* array. The minimum TET value is copied into the first element, and the time at which it was achieved is copied into the second element.

See Also

API functions xPCMaximumTET, xPCAverageTET

Property MinTET of SimulinkRealTime.target

xPCNumLogSamples

Return number of samples in log buffer

Prototype

```
int xPCNumLogSamples(int port);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

Return

The `xPCNumLogSamples` function returns the number of samples in the log buffer. If the function detects an error, it returns `-1`.

Description

The `xPCNumLogSamples` function returns the number of samples in the log buffer. In contrast to `xPCMaxLogSamples`, which returns the maximum number of samples that can be logged (because of buffer size constraints), `xPCNumLogSamples` returns the number of samples actually logged.

Note that the real-time application must be stopped before you get the number.

See Also

API functions `xPCGetStateLog`, `xPCGetOutputLog`, `xPCGetTETLog`, `xPCGetTimeLog`, `xPCMaxLogSamples`

xPCNumLogWraps

Return number of times log buffer wraps

Prototype

```
int xPCNumLogWraps(int port);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

Return

The `xPCNumLogWraps` function returns the number of times the log buffer wraps. If the function detects an error, it returns `-1`.

Description

The `xPCNumLogWraps` function returns the number of times the log buffer wraps.

See Also

API functions `xPCNumLogSamples`, `xPCMaxLogSamples`, `xPCGetStateLog`, `xPCGetOutputLog`, `xPCGetTETLog`, `xPCGetTimeLog`

Property `NumLogWraps` of `SimulinkRealTime.target`

xPCOpenConnection

Open connection to target computer

Prototype

```
void xPCOpenConnection(int port);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

Description

The `xPCOpenConnection` function opens a connection to the target computer whose data is indexed by *`port`*. Before calling this function, set up the target information by calling `xPCRegisterTarget`. A call to either `xPCOpenSerialPort` or `xPCOpenTcpIpPort` can also set up the target information. If the port is already open, calling this function has no effect.

See Also

API functions `xPCOpenTcpIpPort`, `xPCClosePort`, `xPCReOpenPort`, `xPCTargetPing`, `xPCCloseConnection`, `xPCRegisterTarget`

xPCOpenSerialPort

Open RS-232 connection to Simulink Real-Time system

Prototype

```
int xPCOpenSerialPort(int comPort, int baudRate);
```

Arguments

comPort	Index of the COM port to be used (0 is COM1, 1 is COM2, and so forth).
baudRate	*baudRate* must be one of the following values: 1200, 2400, 4800, 9600, 19200, 38400, 57600, or 115200.

Return

The xPCOpenSerialPort function returns the port value for the connection. If the function detects an error, it returns -1.

Description

The xPCOpenSerialPort function initiates an RS-232 connection to a Simulink Real-Time system. It returns the port value for the connection. Be sure to pass this value to all the Simulink Real-Time API functions that require a port value.

If you enter a value of 0 for *baudRate*, this function sets the baud rate to the default value (115200).

Note: RS-232 communication type will be removed in a future release. Use TCP/IP instead.

See Also

API functions `xPCOpenTcpIpPort`, `xPCClosePort`, `xPCReOpenPort`, `xPCTargetPing`, `xPCOpenConnection`, `xPCCloseConnection`, `xPCRegisterTarget`, `xPCDeRegisterTarget`

xPCOpenTcpIpPort

Open TCP/IP connection to Simulink Real-Time system

Prototype

```
int xPCOpenTcpIpPort(const char *ipAddress, const char
*ipPort);
```

Arguments

ipAddress	Enter the IP address of the target as a dotted decimal string. For example, "192.168.0.10".
ipPort	Enter the associated IP port as a string. For example, "22222".

Return

The xPCOpenTcpIpPort function returns a nonnegative integer that you can then use as the port value for a Simulink Real-Time API function that requires it. If this operation fails, this function returns -1.

Description

The xPCOpenTcpIpPort function opens a connection to the TCP/IP location specified by the IP address. It returns a nonnegative integer if it succeeds. Use this integer as the *ipPort* variable in the Simulink Real-Time API functions that require a port value. The global error number is also set, which you can get using xPCGetLastError.

See Also

API functions xPCOpenSerialPort, xPCClosePort, xPCReOpenPort, xPCTargetPing

xPCReboot

Reboot target computer

Prototype

```
void xPCReboot(int port);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

Description

The `xPCReboot` function reboots the target computer. This function returns nothing. This function does not close the connection to the target computer. You should either explicitly close the port or call `xPCReOpenPort` once the target computer has rebooted.

See Also

API function `xPCReOpenPort`

Target object method `SimulinkRealTime.target.reboot`

xPCReOpenPort

Reopen communication channel

Prototype

```
int xPCReOpenPort(int port);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

Return

The `xPCReOpenPort` function returns `0` if it reopens a connection without detecting an error. If the function detects an error, it returns `-1`.

Description

The `xPCReOpenPort` function reopens the communications channel pointed to by *`port`*. The difference between this function and `xPCOpenSerialPort` or `xPCOpenTcpIpPort` is that `xPCReOpenPort` uses the already existing settings, while the other functions need to set up the port.

See Also

API functions `xPCOpenTcpIpPort`, `xPCClosePort`

xPCRegisterTarget

Register target with Simulink Real-Time API library

Prototype

```
int xPCRegisterTarget(int commType, const char *ipAddress,
const char *ipPort, int comPort, int baudRate);
```

Arguments

commType Specify the communication type (TCP/IP or RS-232) between the development and target computers.

> **Note:** RS-232 communication type will be removed in a future release. Use TCP/IP instead.

ipAddress Enter the IP address of the target as a dotted decimal string. For example, `"192.168.0.10"`.

ipPort Enter the associated IP port as a string. For example, `"22222"`.

comPort *comPort* and *baudRate* are as in `xPCOpenSerialPort`.

baudRate The *baudRate* must be one of the following values: `1200`, `2400`, `4800`, `9600`, `19200`, `38400`, `57600`, or `115200`.

Return

The `xPCRegisterTarget` function returns the port number. If the function detects an error, it returns `-1`.

Description

The `xPCRegisterTarget` function works similarly to `xPCOpenSerialPort` and `xPCOpenTcpIpPort`, except that it does not try to open a connection to the target computer. In other words, `xPCOpenSerialPort` or `xPCOpenTcpIpPort` is equivalent to calling `xPCRegisterTarget` with the required parameters, followed by a call to `xPCOpenConnection`.

Use the constants `COMMTYP_TCPIP` and `COMMTYP_RS232` for *`commType`*. If *`commType`* is set to `COMMTYP_RS232`, the function ignores *`ipAddress`* and *`ipPort`*. Analogously, the function ignores *`comPort`* and *`baudRate`* if *`commType`* is set to `COMMTYP_TCPIP`.

If you enter a value of `0` for *`baudRate`*, this function sets the baud rate to the default value (`115200`).

See Also

API functions `xPCDeRegisterTarget`, `xPCOpenTcpIpPort`, `xPCOpenSerialPort`, `xPCClosePort`, `xPCReOpenPort`, `xPCOpenConnection`, `xPCCloseConnection`, `xPCTargetPing`

xPCRemScope

Remove scope

Prototype

```
void xPCRemScope(int port, int scNum);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.

Description

The `xPCRemScope` function removes the scope with number *`scNum`*. Attempting to remove a nonexistent scope causes an error. For a list of existing scopes, see `xPCGetScopes`. Use the `xPCGetScope` function to get the scope number.

See Also

API functions `xPCAddScope`, `xPCScRemSignal`, `xPCGetScopes`

Target object method `SimulinkRealTime.target.remscope`

xPCSaveParamSet

Save parameter values of real-time application

Prototype

```
void xPCSaveParamSet(int port, const char *filename);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
filename	Enter the name of the file to contain the saved parameters.

Description

The `xPCSaveParamSet` function saves the real-time application parameter values in the file *`filename`*. This function saves the file on a local drive of the current target computer. You can later reload these parameters with the `xPCLoadParamSet` function.

You might want to save real-time application parameter values if you change these parameter values while the application is running in `Real-Time` mode. Saving these values enable you to easily recreate real-time application parameter values from a number of application runs.

See Also

API function `xPCLoadParamSet`

xPCScAddSignal

Add signal to scope

Prototype

```
void xPCScAddSignal(int port, int scNum, int sigNum);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.
sigNum	Enter a signal number.

Description

The `xPCScAddSignal` function adds the signal with number *sigNum* to the scope *scNum*. The signal should not already exist in the scope. You can use `xPCScGetSignals` to get a list of the signals already present. Use the function `xPCGetScope` to get the scope number. Use the `xPCGetSignalIdx` function to get the signal number.

See Also

API functions `xPCScRemSignal`, `xPCAddScope`, `xPCRemScope`, `xPCGetScopes`

Scope object methods `SimulinkRealTime.fileScope.addsignal`, `SimulinkRealTime.hostScope.addsignal`, and `SimulinkRealTime.targetScope.addsignal`

xPCScGetAutoRestart

Scope autorestart status

Prototype

```
long xPCScGetAutoRestart(int port, int scNum)
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.

Return

The `xPCScGetAutoRestart` function returns the autorestart flag value of scope *scNum*. If the function detects an error, it returns `-1`.

Description

The `xPCScGetAutoRestart` function gets the autorestart flag value for scope *scNum*. Autorestart flag can be disabled (0) or enabled (1).

See Also

API functions `xPCScSetAutoRestart`

xPCScGetData

Copy scope data to array

Prototype

```
void xPCScGetData(int port, int scNum, int signal_id, int start,
int numsamples, int decimation, double *data);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.
signal_id	Enter a signal number. Enter `-1` to get time stamped data.
start	Enter the first sample from which data retrieval is to start.
numsamples	Enter the number of samples retrieved with a decimation of *decimation*, starting from the *start* value.
decimation	Enter a value such that every *decimation* sample is retrieved in a scope window.
data	The data is available in the array *data*, starting from sample *start*.

Description

The `xPCScGetData` function gets the data used in a scope. Use this function for scopes of type `SCTYPE_HOST`. The scope must be either in state `"Finished"` or in state `"Interrupted"` for the data to be retrievable. (Use the `xPCScGetState` function to check the state of the scope.) The data must be retrieved one signal at a time. The calling function must allocate the space ahead of time to store the scope data. *data* must be an array of doubles, regardless of the data type of the signal to be retrieved. Use the function `xPCScGetSignals` to get the list of signals in the scope for *signal_id*. Use the function `xPCGetScope` to get the scope number for *scNum*.

To get time stamped data, specify `-1` for `signal_id`. From the output, you can then get the number of nonzero elements.

See Also

API functions `xPCGetScope`, `xPCScGetState`, `xPCScGetSignals`

Property `Data` of `SimulinkRealTime.hostScope`

xPCScGetDecimation

Return decimation of scope

Prototype

```
int xPCScGetDecimation(int port, int scNum);
```

Arguments

port	Enter the value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.
scNum	Enter the scope number.

Return

The xPCScGetDecimation function returns the decimation of scope *scNum*. If the function detects an error, it returns -1.

Description

The xPCScGetDecimation function gets the decimation of scope *scNum*. The decimation is a number, N, meaning every Nth sample is acquired in a scope window. Use the xPCGetScope function to get the scope number.

See Also

API function xPCScSetDecimation

Property Decimation of SimulinkRealTime.fileScope, SimulinkRealTime.hostScope, and SimulinkRealTime.targetScope

xPCScGetNumPrePostSamples

Get number of pre- or post-triggering samples before triggering scope

Prototype

```
int xPCScGetNumPrePostSamples(int port, int scNum);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.

Return

The `xPCScGetNumPrePostSamples` function returns the number of samples for pre- or posttriggering for scope *scNum*. If an error occurs, this function returns the minimum integer value (`-2147483647-1`).

Description

The `xPCScGetNumPrePostSamples` function gets the number of samples for pre- or posttriggering for scope *scNum*. A negative number implies pretriggering, whereas a positive number implies posttriggering samples. Use the `xPCGetScope` function to get the scope number.

See Also

API function `xPCScSetNumPrePostSamples`

Property `NumPrePostSamples` of `SimulinkRealTime.fileScope`, `SimulinkRealTime.hostScope`, and `SimulinkRealTime.targetScope`

xPCScGetNumSamples

Get number of samples in one data acquisition cycle

Prototype

```
int xPCScGetNumSamples(int port, int scNum);
```

Arguments

port	Enter the value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.
scNum	Enter the scope number.

Return

The xPCScGetNumSamples function returns the number of samples in the scope *scNum*. If the function detects an error, it returns -1.

Description

The xPCScGetNumSamples function gets the number of samples in one data acquisition cycle for scope *scNum*. Use the xPCGetScope function to get the scope number.

See Also

API function xPCScSetNumSamples

Property NumSamples of SimulinkRealTime.fileScope, SimulinkRealTime.hostScope, and SimulinkRealTime.targetScope

xPCScGetNumSignals

Get number of signals in scope

Prototype

```
int xPCScGetNumSignals(int port, int scNum);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.

Return

The `xPCScGetNumSignals` function returns the number of signals in the scope *scNum*. If the function detects an error, it returns `-1`.

Description

The `xPCScGetNumSignals` function gets the number of signals in the scope *scNum*. Use the `xPCGetScope` function to get the scope number.

See Also

API function `xPCGetScope`

xPCScGetSignalList

Copy list of signals to array

Prototype

```
void xPCScGetSignalList(int port, int scNum, int *data)
```

Arguments

port Value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

scNum Enter the scope number.

data Integer array allocated by the caller as a list containing the signal identifiers.

Description

The `xPCScGetSignals` function gets the list of signals defined for scope *scNum*. The array *data* must be large enough to hold the list of signals. To query the size, use the `xPCScGetNumSignals` function. Use the `xPCGetScope` function to get the scope number.

Note: Use the `xPCScGetSignalList` function instead of the `xPCScGetSignals` function. The `xPCScGetSignals` will be removed in a future release.

xPCScGetSignals

Copy list of signals to array

Prototype

```
void xPCScGetSignals(int port, int scNum, int *data);
```

Arguments

port	Value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.
scNum	Enter the scope number.
data	Integer array allocated by the caller as a list containing the signal identifiers, terminated by -1.

Description

The xPCScGetSignals function gets the list of signals defined for scope *scNum*. You can use the constant MAX_SIGNALS, defined in xpcapiconst.h, as the size of *data*. Use the xPCGetScope function to get the scope number.

Note: This function will be removed in a future release. Use the xPCScGetSignalList function instead.

See Also

API functions xPCScGetData, xPCGetScopes

Scope object property Signals

xPCScGetStartTime

Get start time for last data acquisition cycle

Prototype

```
double xPCScGetStartTime(int port, int scNum);
```

Arguments

port	Enter the value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.
scNum	Enter the scope number.

Return

The xPCScGetStartTime function returns the start time for the last data acquisition cycle of a scope. If the function detects an error, it returns -1.

Description

The xPCScGetStartTime function gets the time at which the last data acquisition cycle for scope *scNum* started. This is only valid for scopes of type SCTYPE_HOST. Use the xPCGetScope function to get the scope number.

See Also

API functions xPCScGetNumSamples, xPCScGetDecimation

xPCScGetState

Get state of scope

Prototype

```
int xPCScGetState(int port, int scNum);
```

Arguments

port	Enter the value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.
scNum	Enter the scope number.

Return

The xPCScGetState function returns the state of scope *scNum*. If the function detects an error, it returns -1.

Description

The xPCScGetState function gets the state of scope *scNum*, or -1 upon error. Use the xPCGetScope function to get the scope number.

Constants to find the scope state, defined in xpcapiconst.h, have the following meanings:

Constant	Value	Description
SCST_WAITTOSTART	0	Scope is ready and waiting to start.
SCST_PREACQUIRING	5	Scope acquires a predefined number of samples before triggering.

Constant	Value	Description
SCST_WAITFORTRIG	1	After a scope is finished with the preacquiring state, it waits for a trigger. If the scope does not preacquire data, it enters the wait for trigger state.
SCST_ACQUIRING	2	Scope is acquiring data. The scope enters this state when it leaves the wait for trigger state.
SCST_FINISHED	3	Scope is finished acquiring data when it has attained the predefined limit.
SCST_INTERRUPTED	4	The user has stopped (interrupted) the scope.

See Also

API functions xPCScStart, xPCScStop

Scope object property Status

xPCScGetTriggerLevel

Get trigger level for scope

Prototype

```
double xPCScGetTriggerLevel(int port, int scNum);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.

Return

The `xPCScGetTriggerLevel` function returns the scope trigger level. If the function detects an error, it returns `-1`.

Description

The `xPCScGetTriggerLevel` function gets the trigger level for scope *scNum*. Use the `xPCGetScope` function to get the scope number.

See Also

API functions `xPCScSetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScGetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`

Property `TriggerLevel` of `SimulinkRealTime.fileScope`, `SimulinkRealTime.hostScope`, and `SimulinkRealTime.targetScope`

xPCScGetTriggerMode

Get trigger mode for scope

Prototype

```
int xPCScGetTriggerMode(int port, int scNum);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.

Return

The `xPCScGetTriggerMode` function returns the scope trigger mode. If the function detects an error, it returns `-1`.

Description

The `xPCScGetTriggerMode` function gets the trigger mode for scope *scNum*. Use the `xPCGetScope` function to get the scope number. Use the constants defined in `xpcapiconst.h` to interpret the trigger mode. These constants include the following:

Constant	Value	Description
`TRIGMD_FREERUN`	0	There is no trigger mode. The scope triggers when it is ready to trigger, regardless of the circumstances.
`TRIGMD_SOFTWARE`	1	Only user intervention can trigger the scope. No other triggering is possible.
`TRIGMD_SIGNAL`	2	The scope is triggered only after a signal has crossed a value.

Constant	Value	Description
TRIGMD_SCOPE	3	The scope is triggered by another scope at the trigger point of the triggering scope, modified by the value of triggerscopesample (see scopedata).

See Also

API functions xPCScSetTriggerLevel, xPCScGetTriggerLevel, xPCScSetTriggerSlope, xPCScGetTriggerSlope, xPCScSetTriggerSignal, xPCScGetTriggerSignal, xPCScSetTriggerScope, xPCScGetTriggerScope, xPCScSetTriggerMode

Methods SimulinkRealTime.fileScope.trigger, SimulinkRealTime.hostScope.trigger, and SimulinkRealTime.targetScope.trigger

Property TriggerMode of SimulinkRealTime.fileScope, SimulinkRealTime.hostScope, and SimulinkRealTime.targetScope

xPCScGetTriggerScope

Get trigger scope

Prototype

```
int xPCScGetTriggerScope(int port, int scNum);
```

Arguments

port	Enter the value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.
scNum	Enter the scope number.

Return

The xPCScGetTriggerScope function returns a trigger scope. If the function detects an error, it returns -1.

Description

The xPCScGetTriggerScope function gets the trigger scope for scope *scNum*. Use the xPCGetScope function to get the scope number.

See Also

API functions xPCScSetTriggerLevel, xPCScGetTriggerLevel, xPCScSetTriggerSlope, xPCScGetTriggerSlope, xPCScSetTriggerSignal, xPCScGetTriggerSignal, xPCScSetTriggerMode, xPCScGetTriggerMode

Property TriggerScope of SimulinkRealTime.fileScope, SimulinkRealTime.hostScope, and SimulinkRealTime.targetScope

xPCScGetTriggerScopeSample

Get sample number for triggering scope

Prototype

```
int xPCScGetTriggerScopeSample(int port, int scNum);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.

Return

The `xPCScGetTriggerScopeSample` function returns a nonnegative integer for a real sample, and `-1` for the special case where triggering is at the end of the data acquisition cycle for a triggering scope. If the function detects an error, it returns `INT_MIN` (`-2147483647-1`).

Description

The `xPCScGetTriggerScopeSample` function gets the number of samples a triggering scope (*scNum*) acquires before starting data acquisition on a second scope. This value is a nonnegative integer for a real sample, and `-1` for the special case where triggering is at the end of the data acquisition cycle for a triggering scope. Use the `xPCGetScope` function to get the scope number for the trigger scope.

See Also

API functions `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`, xPCScGetTriggerSignal, xPCScSetTriggerScope, xPCScGetTriggerScope, xPCScSetTriggerMode, xPCScGetTriggerMode, xPCScSetTriggerScopeSample

Property TriggerSample of SimulinkRealTime.fileScope, SimulinkRealTime.hostScope, and SimulinkRealTime.targetScope

xPCScGetTriggerSignal

Get trigger signal for scope

Prototype

```
int xPCScGetTriggerSignal(int port, int scNum);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.

Return

The `xPCScGetTriggerSignal` function returns the scope trigger signal. If the function detects an error, it returns `-1`.

Description

The `xPCScGetTriggerSignal` function gets the trigger signal for scope *scNum*. Use the `xPCGetScope` function to get the scope number for the trigger scope.

See Also

API functions `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`

Methods `SimulinkRealTime.fileScope.trigger`, `SimulinkRealTime.hostScope.trigger`, and `SimulinkRealTime.targetScope.trigger`

Property `TriggerSignal` of `SimulinkRealTime.fileScope`, `SimulinkRealTime.hostScope`, and `SimulinkRealTime.targetScope`

xPCScGetTriggerSlope

Get trigger slope for scope

Prototype

```
int xPCScGetTriggerSlope(int port, int scNum);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.

Return

The `xPCScGetTriggerSlope` function returns the scope trigger slope. If the function detects an error, it returns `-1`.

Description

The `xPCScGetTriggerSlope` function gets the trigger slope of scope *`scNum`*. Use the `xPCGetScope` function to get the scope number for the trigger scope. Use the constants defined in `xpcapiconst.h` to interpret the trigger slope. These constants have the following meanings:

Constant	Value	Description
`TRIGSLOPE_EITHER`	0	The trigger slope can be either rising or falling.
`TRIGSLOPE_RISING`	1	The trigger slope must be rising when the signal crosses the trigger value.
`TRIGSLOPE_FALLING`	2	The trigger slope must be falling when the signal crosses the trigger value.

See Also

API functions `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScGetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`

Methods `SimulinkRealTime.fileScope.trigger`, `SimulinkRealTime.hostScope.trigger`, and `SimulinkRealTime.targetScope.trigger`

Property `TriggerSlope` of `SimulinkRealTime.fileScope`, `SimulinkRealTime.hostScope`, and `SimulinkRealTime.targetScope`

xPCScGetType

Get type of scope

Prototype

```
int xPCScGetType(int port, int scNum);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.

Return

The `xPCScGetType` function returns the scope type. If the function detects an error, it returns `-1`.

Description

The `xPCScGetType` function gets the type (`SCTYPE_HOST` for host, `SCTYPE_TARGET` for target, or `SCTYPE_FILE` for file) of scope *scNum*. Use the constants defined in `xpcapiconst.h` to interpret the return value. A scope of type `SCTYPE_HOST` is displayed on the development computer while a scope of type `SCTYPE_TARGET` is displayed on the target computer screen. A scope of type `SCTYPE_FILE` is stored on a storage medium. Use the `xPCGetScope` function to get the scope number.

See Also

API functions `xPCAddScope`, `xPCRemScope`

Property `Type` of `SimulinkRealTime.fileScope`, `SimulinkRealTime.hostScope`, and `SimulinkRealTime.targetScope`

xPCScRemSignal

Remove signal from scope

Prototype

```
void xPCScRemSignal(int port, int scNum, int sigNum);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.
sigNum	Enter a signal number.

Description

The `xPCScRemSignal` function removes a signal from the scope with number *scNum*. The scope must already exist, and signal number *sigNum* must exist in the scope. Use `xPCGetScopes` to determine the existing scopes, and use `xPCScGetSignals` to determine the existing signals for a scope. Use this function only when the scope is stopped. Use `xPCScGetState` to check the state of the scope. Use the `xPCGetScope` function to get the scope number.

See Also

API functions `xPCScAddSignal`, `xPCAddScope`, `xPCRemScope`, `xPCGetScopes`, `xPCScGetSignals`, `xPCScGetState`

Scope object methods `SimulinkRealTime.fileScope.remsignal`, `SimulinkRealTime.hostScope.remsignal`, and `SimulinkRealTime.targetScope.remsignal`

xPCScSetAutoRestart

Scope autorestart status

Prototype

```
void xPCScSetAutoRestart(int port, int scNum, int autorestart)
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.
autorestart	Enter value to enable (`1`) or disable (`0`) scope autorestart.

Description

The `xPCScSetAutoRestart` function sets the autorestart flag for scope *scNum* to `0` or `1`. `0` disables the flag, `1` enables it. Use this function only when the scope is stopped.

See Also

API functions `xPCScGetAutoRestart`

xPCScSetDecimation

Set decimation of scope

Prototype

```
void xPCScSetDecimation(int port, int scNum, int decimation);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.
decimation	Enter an integer for the decimation.

Description

The `xPCScSetDecimation` function sets the *decimation* of scope *`scNum`*. The decimation is a number, `N`, meaning every `N`th sample is acquired in a scope window. Use this function only when the scope is stopped. Use `xPCScGetState` to check the state of the scope. Use the `xPCGetScope` function to get the scope number.

See Also

API functions `xPCScGetDecimation`, `xPCScGetState`

Property `Decimation` of `SimulinkRealTime.fileScope`, `SimulinkRealTime.hostScope`, and `SimulinkRealTime.targetScope`

xPCScSetNumPrePostSamples

Set number of pre- or posttriggering samples before triggering scope

Prototype

```
void xPCScSetNumPrePostSamples(int port, int scNum, int prepost);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.
prepost	A negative number means pretriggering, while a positive number means posttriggering. This function can only be used when the scope is stopped.

Description

The `xPCScSetNumPrePostSamples` function sets the number of samples for pre- or posttriggering for scope *scNum* to *prepost*. Use this function only when the scope is stopped. Use `xPCScGetState` to check the state of the scope. Use the `xPCGetScope` function to get the scope number.

See Also

API functions `xPCScGetNumPrePostSamples`, `xPCScGetState`

Property `NumPrePostSamples` of `SimulinkRealTime.fileScope`, `SimulinkRealTime.hostScope`, and `SimulinkRealTime.targetScope`

xPCScSetNumSamples

Set number of samples in one data acquisition cycle

Prototype

```
void xPCScSetNumSamples(int port, int scNum, int samples);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.
samples	Enter the number of samples you want to acquire in one cycle.

Description

The `xPCScSetNumSamples` function sets the number of samples for scope *`scNum`* to *`samples`*. Use this function only when the scope is stopped. Use `xPCScGetState` to check the state of the scope. Use the `xPCGetScope` function to get the scope number.

See Also

API functions `xPCScGetNumSamples`, `xPCScGetState`

Property `NumSamples` of `SimulinkRealTime.fileScope`, `SimulinkRealTime.hostScope`, and `SimulinkRealTime.targetScope`

xPCScSetTriggerLevel

Set trigger level for scope

Prototype

```
void xPCScSetTriggerLevel(int port, int scNum, double level);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.
level	Value for a signal to trigger data acquisition with a scope.

Description

The `xPCScSetTriggerLevel` function sets the trigger level to *level* for scope *scNum*. Use this function only when the scope is stopped. Use `xPCScGetState` to check the state of the scope. Use the `xPCGetScope` function to get the scope number for the trigger scope.

See Also

API functions `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScGetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`, `xPCScGetState`

Property `TriggerLevel` of `SimulinkRealTime.fileScope`, `SimulinkRealTime.hostScope`, and `SimulinkRealTime.targetScope`

xPCScSetTriggerMode

Set trigger mode of scope

Prototype

```
void xPCScSetTriggerMode(int port, int scNum, int mode);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.
mode	Trigger mode for a scope.

Description

The `xPCScSetTriggerMode` function sets the trigger mode of scope *`scNum`* to *`mode`*. Use this function only when the scope is stopped. Use `xPCScGetState` to check the state of the scope. Use the `xPCGetScopes` function to get a list of scopes.

Use the constants defined in `xpcapiconst.h` to interpret the trigger mode:

Constant	Value	Description
`TRIGMD_FREERUN`	0	There is no trigger mode. The scope triggers when it is ready to trigger, regardless of the circumstances. This is the default.
`TRIGMD_SOFTWARE`	1	Only user intervention can trigger the scope. No other triggering is possible.
`TRIGMD_SIGNAL`	2	The scope is triggered only after a signal has crossed a value.
`TRIGMD_SCOPE`	3	The scope is triggered by another scope at the trigger point of the triggering scope, modified by the value of `triggerscopesample` (see `scopedata`).

See Also

API functions `xPCGetScopes`, `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScGetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScGetTriggerMode`, `xPCScGetState`

Methods `SimulinkRealTime.fileScope.trigger`, `SimulinkRealTime.hostScope.trigger`, and `SimulinkRealTime.targetScope.trigger`

Property `TriggerMode` of `SimulinkRealTime.fileScope`, `SimulinkRealTime.hostScope`, and `SimulinkRealTime.targetScope`

xPCScSetTriggerScope

Select scope to trigger another scope

Prototype

```
void xPCScSetTriggerScope(int port, int scNum, int trigScope);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.
trigScope	Enter the scope number of the scope used for a trigger.

Description

The `xPCScSetTriggerScope` function sets the trigger scope of scope *scNum* to *trigScope*. This function can only be used when the scope is stopped. Use `xPCScGetState` to check the state of the scope. Use the `xPCGetScopes` function to get a list of scopes.

The scope type can be `SCTYPE_HOST`, `SCTYPE_TARGET`, or `SCTYPE_FILE`.

See Also

API functions `xPCGetScopes`, `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScGetTriggerSignal`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`, `xPCScGetState`

Property `TriggerScope` of `SimulinkRealTime.fileScope`, `SimulinkRealTime.hostScope`, and `SimulinkRealTime.targetScope`

xPCScSetTriggerScopeSample

Set sample number for triggering scope

Prototype

```
void xPCScSetTriggerScopeSample(int port, int scNum, int
trigScSamp);
```

Arguments

port Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

scNum Enter the scope number.

trigScSamp Enter a nonnegative integer for the number of samples acquired by the triggering scope before starting data acquisition on a second scope.

Description

The `xPCScSetTriggerScopeSample` function sets the number of samples (*trigScSamp*) a triggering scope acquires before it triggers a second scope (*scNum*). Use the `xPCGetScopes` function to get a list of scopes.

For meaningful results, set *trigScSamp* between `-1` and (*nSamp*`-1`). *nSamp* is the number of samples in one data acquisition cycle for the triggering scope. If you specify too large a value, the scope is never triggered.

If you want to trigger a second scope at the end of a data acquisition cycle for the triggering scope, enter a value of `-1` for *trigScSamp*.

See Also

API functions `xPCGetScopes`, `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScGetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`, `xPCScGetTriggerScopeSample`

Property `TriggerSample` of `SimulinkRealTime.fileScope`, `SimulinkRealTime.hostScope`, and `SimulinkRealTime.targetScope`

xPCScSetTriggerSignal

Select signal to trigger scope

Prototype

```
void xPCScSetTriggerSignal(int port, int scNum, int trigSig);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.
trigSig	Enter a signal number.

Description

The `xPCScSetTriggerSignal` function sets the trigger signal of scope *scNum* to *trigSig*. The trigger signal *trigSig* must be one of the signals in the scope. Use this function only when the scope is stopped. You can use `xPCScGetSignals` to get the list of signals in the scope. Use `xPCScGetState` to check the state of the scope. Use the `xPCGetScopes` function to get a list of scopes.

See Also

API functions `xPCGetScopes`, `xPCScGetState`, `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScGetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`

Property `TriggerSignal` of `SimulinkRealTime.fileScope`, `SimulinkRealTime.hostScope`, and `SimulinkRealTime.targetScope`

xPCScSetTriggerSlope

Set slope of signal that triggers scope

Prototype

```
void xPCScSetTriggerSlope(int port, int scNum, int trigSlope);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.
trigSlope	Enter the slope mode for the signal that triggers the scope.

Description

The `xPCScSetTriggerSlope` function sets the trigger slope of scope *scNum* to *trigSlope*. Use this function only when the scope is stopped. Use `xPCScGetState` to check the state of the scope. Use the `xPCGetScopes` function to get a list of scopes.

Use the constants defined in `xpcapiconst.h` to set the trigger slope:

Constant	Value	Description
`TRIGSLOPE_EITHER`	0	The trigger slope can be either rising or falling.
`TRIGSLOPE_RISING`	1	The trigger signal value must be rising when it crosses the trigger value.
`TRIGSLOPE_FALLING`	2	The trigger signal value must be falling when it crosses the trigger value.

See Also

API functions xPCGetScopes, xPCScSetTriggerLevel, xPCScGetTriggerLevel, xPCScGetTriggerSlope, xPCScSetTriggerSignal, xPCScGetTriggerSignal, xPCScSetTriggerScope, xPCScGetTriggerScope, xPCScSetTriggerMode, xPCScGetTriggerMode, xPCScGetState

Property TriggerSlope of SimulinkRealTime.fileScope, SimulinkRealTime.hostScope, and SimulinkRealTime.targetScope

xPCScSoftwareTrigger

Set software trigger of scope

Prototype

```
void xPCScSoftwareTrigger(int port, int scNum);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.

Description

The `xPCScSoftwareTrigger` function triggers scope *`scNum`*. The scope must be in the state `Waiting for trigger` for this function to succeed. Use `xPCScGetState` to check the state of the scope. Use the `xPCGetScopes` function to get a list of scopes.

Regardless of the trigger mode setting, you can use `xPCScSoftwareTrigger` to force a trigger. In trigger mode `Software`, this function is the only way to trigger the scope.

See Also

API functions `xPCGetScopes`, `xPCScGetState`, `xPCIsScFinished`

Methods `SimulinkRealTime.fileScope.trigger`, `SimulinkRealTime.hostScope.trigger`, and `SimulinkRealTime.targetScope.trigger`

Property `TriggerMode` of `SimulinkRealTime.fileScope`, `SimulinkRealTime.hostScope`, and `SimulinkRealTime.targetScope`

xPCScStart

Start data acquisition for scope

Prototype

```
void xPCScStart(int port, int scNum);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.

Description

The `xPCScStart` function starts or restarts the data acquisition of scope *`scNum`*. If the scope does not have to preacquire samples, it enters the `Waiting for Trigger` state. The scope must be in state `Waiting to Start`, `Finished`, or `Interrupted` for this function to succeed. Call `xPCScGetState` to check the state of the scope or, for host scopes that are already started, call `xPCIsScFinished`. Use the `xPCGetScopes` function to get a list of scopes.

See Also

API functions `xPCGetScopes`, `xPCScGetState`, `xPCScStop`, `xPCIsScFinished`

Scope object method `SimulinkRealTime.fileScope.start`, `SimulinkRealTime.hostScope.start`, `SimulinkRealTime.targetScope.start`

xPCScStop

Stop data acquisition for scope

Prototype

```
void xPCScStop(int port, int scNum);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.

Description

The `xPCScStop` function stops the scope *scNum*. This sets the scope to the `"Interrupted"` state. The scope must be running for this function to succeed. Use `xPCScGetState` to determine the state of the scope. Use the `xPCGetScopes` function to get a list of scopes.

See Also

API functions `xPCGetScopes`, `xPCScStart`, `xPCScGetState`

Scope object methods `SimulinkRealTime.fileScope.stop`, `SimulinkRealTime.hostScope.stop`, `SimulinkRealTime.targetScope.stop`

xPCSetEcho

Turn message display on or off

Prototype

```
void xPCSetEcho(int port, int mode);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
mode	Valid values are
	`0` Turns the display off
	`1` Turns the display on

Description

On the target computer screen, the `xPCSetEcho` function sets the message display on the target computer on or off. You can change the mode only when the real-time application is stopped. When you turn the message display off, the message screen no longer updates. Existing messages remain on the screen as they were.

See Also

API function `xPCGetEcho`

xPCSetLastError

Set last error to specific string constant

Prototype

```
void xPCSetLastError(int error);
```

Arguments

error	Specify the string constant for the error.

Description

The `xPCSetLastError` function sets the global error constant returned by `xPCGetLastError` to *`error`*. This is useful only to set the string constant to `ENOERR`, indicating no error was found.

See Also

API functions `xPCGetLastError`, `xPCErrorMsg`

xPCSetLoadTimeOut

Change initialization timeout value between development and target computers

Prototype

```
void xPCSetLoadTimeOut(int port, int timeOut);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
timeOut	Enter the new communication timeout value.

Description

The `xPCSetLoadTimeOut` function changes the timeout value for communication between the development and target computers. The *`timeOut`* value is the time a Simulink Real-Time API function waits for the communication to complete before returning. It enables you to set the number of communication attempts to be made before signaling a timeout.

For example, the function `xPCLoadApp` waits to check whether the model initialization for a new application is complete before returning. When a new real-time application is loaded onto the target computer, the function `xPCLoadApp` waits for a certain time to check whether the model initialization is complete before returning. If the model initialization is incomplete within the allotted time, `xPCLoadApp` returns a timeout error.

By default, `xPCLoadApp` checks for target readiness for up to 5 seconds. However, for larger models or models requiring longer initialization (for example, models with thermocouple boards), the default might not be long enough and a spurious timeout can be generated. Other functions that communicate with the target computer will wait for *`timeOut`* seconds before declaring a timeout event.

See Also

API functions `xPCGetLoadTimeOut`, `xPCLoadApp`, `xPCUnloadApp`

xPCSetLogMode

Set logging mode and increment value of scope

Prototype

```
void xPCSetLogMode(int port, lgmode logging_data);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
logging_data	Logging mode and increment value.

Description

The `xPCSetLogMode` function sets the logging mode and increment to the values set in *logging_data*. See the structure `lgmode` for more details.

See Also

API function `xPCGetLogMode`

API structure `lgmode`

Property `LogMode` of `SimulinkRealTime.target`

xPCSetParam

Change value of parameter

Prototype

```
void xPCSetParam(int port, int paramIdx, const double *paramValue);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
paramIdx	Parameter index.
paramValue	Vector of doubles, assumed to be the size required by the parameter type

Description

The `xPCSetParam` function sets the parameter *`paramIdx`* to the value in *`paramValue`*. For matrices, *`paramValue`* should be a vector representation of the matrix in column-major format. Although *`paramValue`* is a vector of doubles, the function converts the values to the expected data types (using truncation) before setting them.

See Also

API functions `xPCGetParamDims`, `xPCGetParamIdx`, `xPCGetParam`

xPCSetSampleTime

Change real-time application sample time

Prototype

```
void xPCSetSampleTime(int port, double ts);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
ts	Sample time for the real-time application.

Description

The `xPCSetSampleTime` function sets the sample time, in seconds, of the real-time application to *ts*. Use this function only when the application is stopped.

See Also

API function `xPCGetSampleTime`

Property `SampleTime` of `SimulinkRealTime.target`

xPCSetScope

Set properties of scope

Prototype

```
void xPCSetScope(int port, scopedata state);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
state	Enter a structure of type `scopedata`.

Description

Note: The `xPCSetScope` function will be removed in a future release. Use the `xPCScSet`*`ScopePropertyName`* functions to access property values instead. For example, to set the number of samples to acquire in one data acquisition cycle, use `xPCScSetNumSamples`.

The `xPCSetScope` function sets the properties of a scope using a *`state`* structure of type `scopedata`. Set the properties you want to set for the scope. You can set several properties at the same time. For convenience, call the function `xPCGetScope` first to populate the structure with the current values. You can then change the desired values. Use this function only when the scope is stopped. Use `xPCScGetState` to determine the state of the scope.

See Also

API functions `xPCGetScope`, `xPCScGetState`, `scopedata`

xPCSetStopTime

Change real-time application stop time

Prototype

```
void xPCSetStopTime(int port, double tfinal);
```

Arguments

port	Enter the value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.
tfinal	Enter the stop time, in seconds.

Description

The xPCSetStopTime function sets the stop time of the real-time application to the value in *tfinal*. The real-time application will run for this number of seconds before stopping. Set *tfinal* to -1.0 to set the stop time to infinity.

See Also

API function xPCGetStopTime

Property StopTime of SimulinkRealTime.target

xPCStartApp

Start real-time application

Prototype

```
void xPCStartApp(int port);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

Description

The `xPCStartApp` function starts the real-time application loaded on the target computer.

See Also

API function `xPCStopApp`

Target object method `SimulinkRealTime.target.start`

xPCStopApp

Stop real-time application

Prototype

```
void xPCStopApp(int port);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

Description

The `xPCStopApp` function stops the real-time application loaded on the target computer. The real-time application remains loaded and the parameter changes you made remain intact. If you want to stop and unload an application, use `xPCUnloadApp`.

See Also

API functions `xPCStartApp`, `xPCUnloadApp`

Target object method `SimulinkRealTime.target.stop`

xPCTargetPing

Ping target computer

Prototype

```
int xPCTargetPing(int port);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

Return

The `xPCTargetPing` function does not return an error status. This function returns `1` if the target responds. If the target computer does not respond, the function returns `0`.

Description

The `xPCTargetPing` function pings the target computer and returns `1` or `0` depending on whether the target responds or not. This function returns an error string constant only when there is an error in the input parameter (for example, the port number is invalid or *port* is not open). Other errors, such as the inability to connect to the target, are ignored.

If you are using TCP/IP, note that `xPCTargetPing` will cause the target computer to close the TCP/IP connection. You can use `xPCOpenConnection` to reconnect. You can also use this `xPCTargetPing` feature to close the target computer connection in the event of an aborted TCP/IP connection (for example, if the program running on your development computer has a fatal error).

See Also

API functions `xPCOpenConnection`, `xPCOpenSerialPort`, `xPCOpenTcpIpPort`, `xPCClosePort`

xPCTgScGetGrid

Get status of grid line for particular scope

Prototype

```
int xPCTgScGetGrid(int port, int scNum);
```

Arguments

port	Enter the value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.
scNum	Enter the scope number.

Return

Returns the status of the grid for a scope of type SCTYPE_TARGET. If the function detects an error, it returns -1.

Description

The xPCTgScGetGrid function gets the state of the grid lines for scope *scNum* (which must be of type SCTYPE_TARGET). A return value of 1 implies grid on, while 0 implies grid off. Note that when the scope mode is set to SCMODE_NUMERICAL, the grid is not drawn even when the grid mode is set to 1.

Tip

- Use xPCTgScSetMode and xPCTgScGetMode to set and retrieve the scope mode.
- Use xPCGetScopes to get a list of scopes.

See Also

API functions `xPCGetScopes`, `xPCTgScSetGrid`, `xPCTgScSetViewMode`, `xPCTgScGetViewMode`, `xPCTgScSetMode`, `xPCTgScGetMode`, `xPCTgScSetYLimits`, `xPCTgScGetYLimits`

xPCTgScGetMode

Get scope mode for displaying signals

Prototype

```
int xPCTgScGetMode(int port, int scNum);
```

Arguments

port Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

scNum Enter the scope number.

Return

The `xPCTgScGetMode` function returns the value corresponding to the scope mode. The possible values are

- `SCMODE_NUMERICAL = 0`
- `SCMODE_REDRAW = 1`
- `SCMODE_SLIDING = 2`
- `SCMODE_ROLLING = 3`

If this function detects an error, it returns `-1`.

Description

The `xPCTgScGetMode` function gets the mode (`SCMODE_NUMERICAL`, `SCMODE_REDRAW`, `SCMODE_SLIDING`, `SCMODE_ROLLING`) of the scope *scNum*, which must be of type `SCTYPE_TARGET`. Use the `xPCGetScopes` function to get a list of scopes.

See Also

API functions xPCGetScopes, xPCTgScSetGrid, xPCTgScGetGrid, xPCTgScSetViewMode, xPCTgScGetViewMode, xPCTgScSetMode, xPCTgScSetYLimits, xPCTgScGetYLimits

Property DisplayMode of SimulinkRealTime.fileScope, SimulinkRealTime.hostScope, and SimulinkRealTime.targetScope

xPCTgScGetViewMode

Get view mode for target computer display

Prototype

```
int xPCTgScGetViewMode(int port);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

Return

The `xPCTgScGetViewMode` function returns the view mode for the target computer screen. If the function detects an error, it returns `-1`.

Description

The `xPCTgScGetViewMode` function gets the view (zoom) mode for the target computer display. If the returned value is not zero, the number is that of the scope currently displayed on the screen. If the value is `0`, then all defined scopes are displayed on the target computer screen, but no scopes are in focus (all scopes are unzoomed).

See Also

API functions `xPCGetScopes`, `xPCTgScSetGrid`, `xPCTgScGetGrid`, `xPCTgScSetViewMode`, `xPCTgScSetMode`, `xPCTgScGetMode`, `xPCTgScSetYLimits`, `xPCTgScGetYLimits`

Property `ViewMode` of `SimulinkRealTime.target`

xPCTgScGetYLimits

Copy *y*-axis limits for scope to array

Prototype

```
void xPCTgScGetYLimits(int port, int scNum, double *limits);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.
limits	The first element of the array is the lower limit while the second element is the upper limit.

Description

The `xPCTgScGetYLimits` function gets and copies the upper and lower limits for a scope of type `SCTYPE_TARGET` and with scope number *`scNum`*. The limits are stored in the array *`limits`*. If both elements are zero, the limits are autoscaled. Use the `xPCGetScopes` function to get a list of scopes.

See Also

API functions `xPCGetScopes`, `xPCTgScSetGrid`, `xPCTgScGetGrid`, `xPCTgScSetViewMode`, `xPCTgScGetViewMode`, `xPCTgScSetMode`, `xPCTgScGetMode`, `xPCTgScSetYLimits`

Property `Ylimit` of `SimulinkRealTime.targetScope`

xPCTgScSetGrid

Set grid mode for scope

Prototype

```
void xPCTgScSetGrid(int port, int scNum, int grid);
```

Arguments

port	Enter the value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.
scNum	Enter the scope number.
grid	Enter a grid value.

Description

The xPCTgScSetGrid function sets the grid of a scope of type SCTYPE_TARGET and scope number *scNum* to *grid*. If *grid* is 0, the grid is off. If *grid* is 1, the grid is on and grid lines are drawn on the scope window. When the drawing mode of scope *scNum* is set to SCMODE_NUMERICAL, the grid is not drawn even when the grid mode is set to 1. Use the xPCGetScopes function to get a list of scopes.

See Also

API functions xPCGetScopes, xPCTgScGetGrid, xPCTgScSetViewMode, xPCTgScGetViewMode, xPCTgScSetMode, xPCTgScGetMode, xPCTgScSetYLimits, xPCTgScGetYLimits

Scope object property Grid

xPCTgScSetMode

Set display mode for scope

Prototype

```
void xPCTgScSetMode(int port, int scNum, int mode);
```

Arguments

port	Enter the value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.
scNum	Enter the scope number.
mode	Enter the value for the mode.

Description

The xPCTgScSetMode function sets the mode of a scope of type SCTYPE_TARGET and scope number *scNum* to *mode*. You can use one of the following constants for *mode*:

- SCMODE_NUMERICAL = 0
- SCMODE_REDRAW = 1
- SCMODE_SLIDING = 2
- SCMODE_ROLLING = 3

Use the xPCGetScopes function to get a list of scopes.

See Also

API functions xPCGetScopes, xPCTgScSetGrid, xPCTgScGetGrid, xPCTgScSetViewMode, xPCTgScGetViewMode, xPCTgScGetMode, xPCTgScSetYLimits, xPCTgScGetYLimits

Property `DisplayMode` of `SimulinkRealTime.targetScope`

xPCTgScSetViewMode

Set view mode for scope

Prototype

```
void xPCTgScSetViewMode(int port, int scNum);
```

Arguments

`port` Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

`scNum` Enter the scope number.

Description

The `xPCTgScSetViewMode` function sets the target computer screen to display one scope with scope number *`scNum`*. If you set *`scNum`* to `0`, the target computer screen displays all the defined scopes. Use the `xPCGetScopes` function to get a list of scopes.

See Also

API functions `xPCGetScopes`, `xPCTgScSetGrid`, `xPCTgScGetGrid`, `xPCTgScGetViewMode`, `xPCTgScSetMode`, `xPCTgScGetMode`, `xPCTgScSetYLimits`, `xPCTgScGetYLimits`

Property `ViewMode` of `SimulinkRealTime.target`

xPCTgScSetYLimits

Set *y*-axis limits for scope

Prototype

```
void xPCTgScSetYLimits(int port, int scNum, const double *Ylimits);
```

Arguments

port	Enter the value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.
scNum	Enter the scope number.
Ylimits	Enter a two-element array.

Description

The xPCTgScSetYLimits function sets the *y*-axis limits for a scope with scope number *scNum* and type SCTYPE_TARGET to the values in the double array *Ylimits*. The first element is the lower limit, and the second element is the upper limit. Set both limits to 0.0 to specify autoscaling. Use the xPCGetScopes function to get a list of scopes.

See Also

API functions xPCGetScopes, xPCTgScSetGrid, xPCTgScGetGrid, xPCTgScSetViewMode, xPCTgScGetViewMode, xPCTgScSetMode, xPCTgScGetMode, xPCTgScGetYLimits

Property Ylimit of SimulinkRealTime.targetScope

xPCUnloadApp

Unload real-time application

Prototype

```
void xPCUnloadApp(int port);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

Description

The `xPCUnloadApp` function stops the current real-time application, removes it from the target computer memory, and resets the target computer in preparation for receiving a new real-time application. The function `xPCLoadApp` calls this function before loading a new real-time application.

See Also

API function `xPCLoadApp`

Target object methods `SimulinkRealTime.target.load`, `SimulinkRealTime.target.unload`

7

MATLAB API

fc422mexcalcbits

Calculate parameter values for Fastcom 422/2-PCI board

Syntax

```
[a,b] = fc422mexcalcbits(frequency)
[a,b,df] = fc422mexcalcbits(frequency)
```

Description

`[a,b] = fc422mexcalcbits(frequency)` accepts a baud rate and converts this value into values for the parameter **Clocks Bits** of the Fastcom® 422/2-PCI driver clock.

`[a,b,df] = fc422mexcalcbits(frequency)` accepts a baud rate and converts this value into a vector containing:

- Values for the parameter **Clocks Bits** of the Fastcom 422/2-PCI driver block.
- The actual baud rate that is created by the **Clocks Bits** parameters.

Examples

Clocks Bits Values

```
[a,b] = fc422mexcalcbits(30e3)
```

```
a =

     2111792


b =

    23
```

Clocks Bits Values with Actual Result

```
[a,b,df] = fc422mexcalcbits(1.49e6)
```

```
a =

     3805896


b =

    23


df =

   1.4901e+06
```

Input Arguments

`frequency` — Baud rate for the board, in units of baud/second

positive-valued scalar

The baud rate must be between `30e3` and `1.5e6`. This limitation is a hardware limitation of the clock circuit.

Example: `30e3`

Data Types: `double`

Output Arguments

`[a,b]` — Values for driver block parameter

vector of scalars

`[a,b,df]` — Values for driver block parameter and actual baud rate that results

vector of scalars

- `a,b` – Values for the driver block parameter.
- `df` – The actual baud rate that is created by the driver block parameter. The clock circuit has limited resolution and is unable to perfectly match an arbitrary frequency.

Introduced in R2014a

macaddr

Convert string-based MAC address to vector-based address

Syntax

```
macaddr(MAC_address)
```

Description

`macaddr(MAC_address)` converts a string-based MAC address to a vector-based MAC address.

Examples

Simple

```
macaddr('01:23:45:67:89:ab')
```

```
ans =

     1    35    69   103   137   171
```

Input Arguments

`MAC_address` — MAC address to be converted

delimited string

The value is entered as a string comprised of six colon-delimited fields of two-digit hexadecimal numbers.

Example: '01:23:45:67:89:ab'

Data Types: `char`

See Also

"Model-Based Ethernet Communications"

Introduced in R2014a

profile_xpc

Collect profiling data

Syntax

```
profData = profile_xpc(profileInfo)
```

Description

`profData = profile_xpc(profileInfo)` collects and displays execution profiling data from a target computer that is running a suitably configured application. By default, it displays an execution profile plot and a code execution profiling report.

To configure a model for execution profiling, check the **Measure task execution time** option in the **Verification** tab of the Code Generation dialog box.

Examples

Concurrent Execution Example

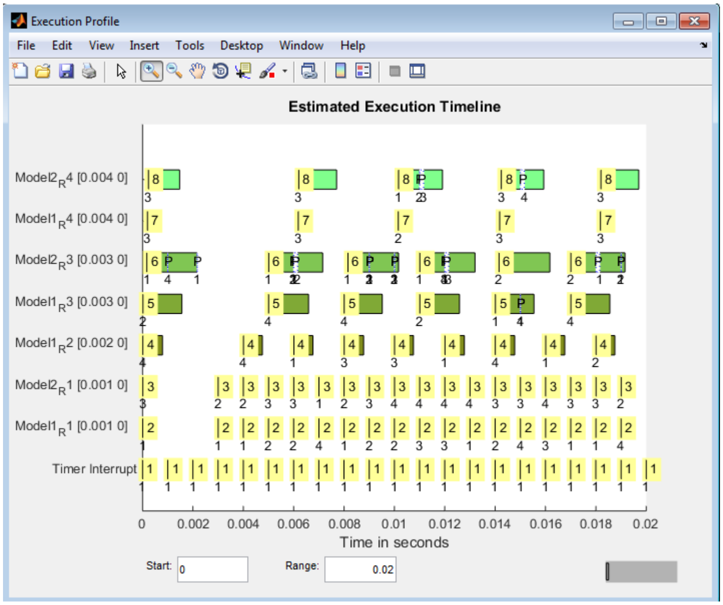
Profile the concurrent execution model `dxpcmds6t` using default settings on a multicore target computer.

Configure model `dxpcmds6t` for profiling. Build, download, and execute the model.

Profile the real-time application execution.

```
profileInfo.modelname = 'dxpcmds6t.mdl';
profData = profile_xpc(profileInfo);
```

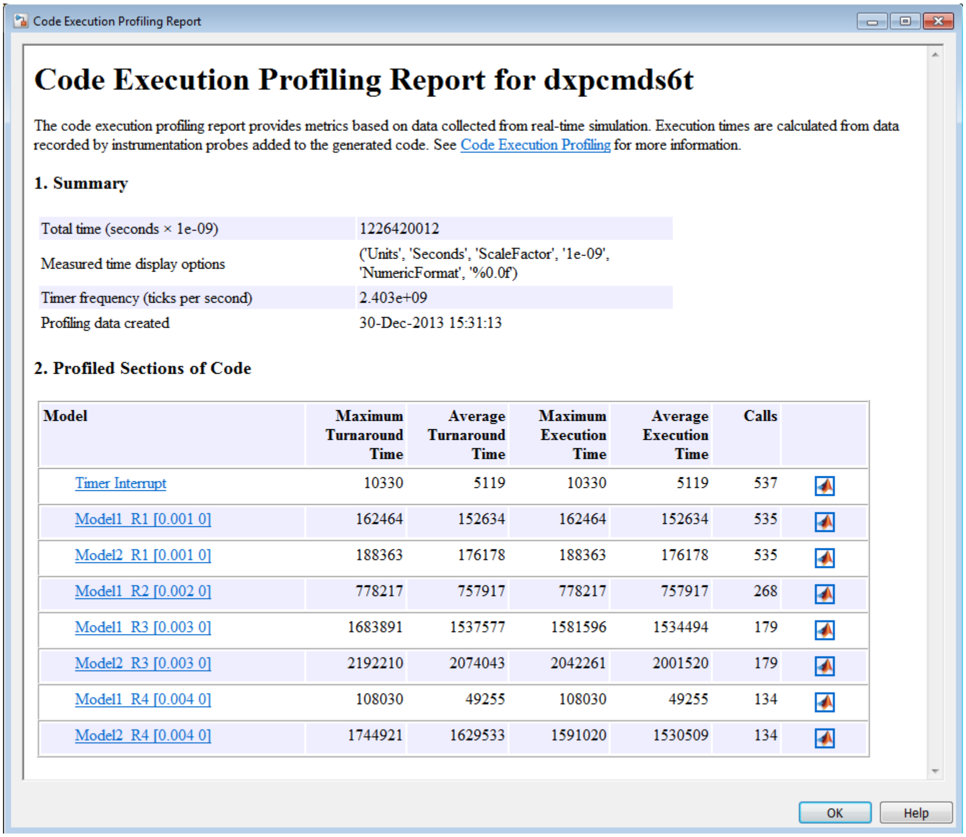
The Execution Profile plot shows the allocation of execution cycles across the four processors, indicated by the colored horizontal bars.



Start: 0

Range: 0.02

The Code Execution Profiling Report displays model execution profile results for each task.



Code Execution Profiling Report for dxpcmds6t

The code execution profiling report provides metrics based on data collected from real-time simulation. Execution times are calculated from data recorded by instrumentation probes added to the generated code. See Code Execution Profiling for more information.

1. Summary

Total time (seconds × 1e-09)	1226420012
Measured time display options	('Units', 'Seconds', 'ScaleFactor', '1e-09', 'NumericFormat', '%0.0f')
Timer frequency (ticks per second)	2.403e+09
Profiling data created	30-Dec-2013 15:31:13

2. Profiled Sections of Code

Model	Maximum Turnaround Time	Average Turnaround Time	Maximum Execution Time	Average Execution Time	Calls
Timer Interrupt	10330	5119	10330	5119	537
Model1_R1 [0.001 0]	162464	152634	162464	152634	535
Model2_R1 [0.001 0]	188363	176178	188363	176178	535
Model1_R2 [0.002 0]	778217	757917	778217	757917	268
Model1_R3 [0.003 0]	1683891	1537577	1581596	1534494	179
Model2_R3 [0.003 0]	2192210	2074043	2042261	2001520	179
Model1_R4 [0.004 0]	108030	49255	108030	49255	134
Model2_R4 [0.004 0]	1744921	1629533	1591020	1530509	134

Profile Data	Description
Maximum turnaround time	Longest time between when the task starts and finishes. This time includes task preemptions (interrupts).
Average turnaround time	Average time between when the task starts and finishes. This time includes task preemptions (interrupts).
Maximum execution time	Longest time between when the task starts and finishes. This time does not include task preemptions (interrupts).
Average execution time	Average time between when the task starts and finishes. This time does not include task preemptions (interrupts).
Calls	Number of times the generated code section is called.

To display the profile data for the generated code section, click the Membrane icon in the Coder Execution Profiling Report.

- "Configure Real-Time Application for Profiling"
- "Generate Real-Time Application Execution Profile"

Input Arguments

`profileInfo` — Profile configuration information

structure

Profile configuration data, consisting of the following fields:

`rawdataonhost` — Flag specifying whether the raw data is on development or target computer

`0` (default) | `1`

- `0` — The raw data file `xPCTrace.csv` is on the target computer. Transfer the file from the target computer to the host.
- `1` — The raw data file `xPCTrace.csv` is in the current folder on the development computer.

Data Types: `double`

`modelname` — Name of the model to be profiled

usrname

The name can include the model file extension.

Data Types: `char`

`noplot` — Flag suppressing execution profile plot

`0` (default) | `1`

- `0` — Display the execution profile plot on the development computer monitor.
- `1` — Do not display the execution profile plot on the development computer monitor.

Data Types: `double`

`noreport` — Flag suppressing code execution profiling report

`0` (default) | `1`

- `0` — Display the code execution profiling report on the development computer monitor.
- `1` — Do not display the code execution profiling report on the development computer monitor.

Data Types: `double`

Output Arguments

`profData` — Profile results data

structure

Profile results data stored in an object of type `coder.profile.ExecutionTime`.

`TimerTicksPerSecond` — Number of seconds per timer tick

double

Scales the execution time tick.

`Sections` — Array of results data for profiled code sections

array

Each array item is an object of type `coder.profile.ExecutionTimeSection`.

See Also

`Sections` | `TimerTicksPerSecond`

Introduced in R2014a

slrt

Create object to manage target computer

Syntax

```
target_object = slrt
target_object = slrt(target_name)
```

Description

`target_object = slrt` constructs a target object representing the default target computer.

When MATLAB evaluates the return value on the development computer, it attempts to connect to the target computer. If the attempt succeeds, MATLAB prints `Connected = Yes`, followed by the status of the application running on the target computer. If the attempt fails, MATLAB waits until the connection times out, and then prints `Connected = No`. To avoid the timeout delay, verify that the target computer is operational and connected to the development computer, or suppress output with a terminating semicolon.

`target_object = slrt(target_name)` constructs a target object representing the target computer designated by `target_name`.

Examples

Default Target Computer

Create a target object that communicates with the default target computer. Report the status of the default target computer. In this case, the target computer is connected to the development computer and is executing the loader.

```
target_object = slrt
```

```
Target: TargetPC1
   Connected            = Yes
   Application          = loader
```

Specific Target Computer

Create a target object that communicates with target computer `TargetPC1`. Report the status of the target computer. In this case, the target computer is not connected to the development computer.

```
target_object = slrt('TargetPC1')
```

```
Target: TargetPC1
   Connected            = No
```

Input Arguments

`target_name` — Name assigned to target computer
string

Example: `'TargetPC1'`

Data Types: `char`

Output Arguments

`target_object` — Object representing target computer
object variable

Object of type `SimulinkRealTime.target` that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required the Ethernet link settings.

Example: `tg`

Data Types: `struct`

See Also

`SimulinkRealTime.target` | Target Settings Properties

Introduced in R2014a

slrtbench

Benchmark Simulink Real-Time models on target computer

Syntax

```
slrtbench
slrtbench benchmark
slrtbench benchmark -reboot
slrtbench benchmark -cleanup
slrtbench benchmark -verbose
slrtbench benchmark -reboot -cleanup -verbose

expected_results = slrtbench()
current_results = slrtbench(benchmark,___)
```

Description

`slrtbench` benchmarks the real-time execution performance of Simulink Real-Time applications on your target computer. It compares the result to stored benchmark results from other computers.

Benchmark execution includes generating benchmark models, building and downloading Simulink Real-Time applications, searching for the minimal achievable sample time, and displaying results.

`slrtbench` without an argument displays representative results for benchmarks run on various target computers with various compiler versions. Display includes:

- Relative Performance — Bar graph containing the computers tested, ranked by relative performance.
- Minimal achievable sample times in µs — Table containing, for each target computer tested, the minimal achievable sample time for the benchmarks, in microseconds.
- Target Information — Technical information about the target computers benchmarked.

Depending upon the value of `benchmark`, `slrtbench benchmark` produces different outputs:

- `slrtbench this` displays benchmark results your target computer, compared with the representative benchmark results for other target computers:

  - Relative Performance — Bar graph containing the computers tested, ranked by relative performance.
  - Minimal achievable sample times in µs — Table containing, for each target computer tested, the minimal achievable sample time for the benchmarks, in microseconds.
  - Target Information — Technical information about the target computers benchmarked.

  The entry for your target computer is highlighted.
- `slrtbench benchmark` prints the benchmark name, the number of blocks, the model build time in seconds, the execution time in seconds, and the minimal achievable sample time in microseconds in the Command Window.

`slrtbench benchmark -reboot` runs the benchmark, then restarts the target computer.

`slrtbench benchmark -cleanup` runs the benchmark, plots or prints benchmark results, and deletes the build files.

`slrtbench benchmark -verbose` prints build output, runs the benchmark, and plots or prints benchmark results.

`slrtbench benchmark -reboot -cleanup -verbose` prints build output, restarts the target computer, deletes build files, and plots or prints results.

You can add zero or more of these control arguments in arbitrary order.

`expected_results = slrtbench()` returns the benchmark results for the five predefined benchmarks in a structure array.

Depending upon the value of `benchmark`, `current_results = slrtbench( benchmark, ___ )` returns different results:

- `slrtbench('this')` returns the benchmark results for the predefined benchmarks in a structure array.
- `slrtbench(benchmark)` returns the benchmark results for the specified model in a structure.

Examples

slrtbench

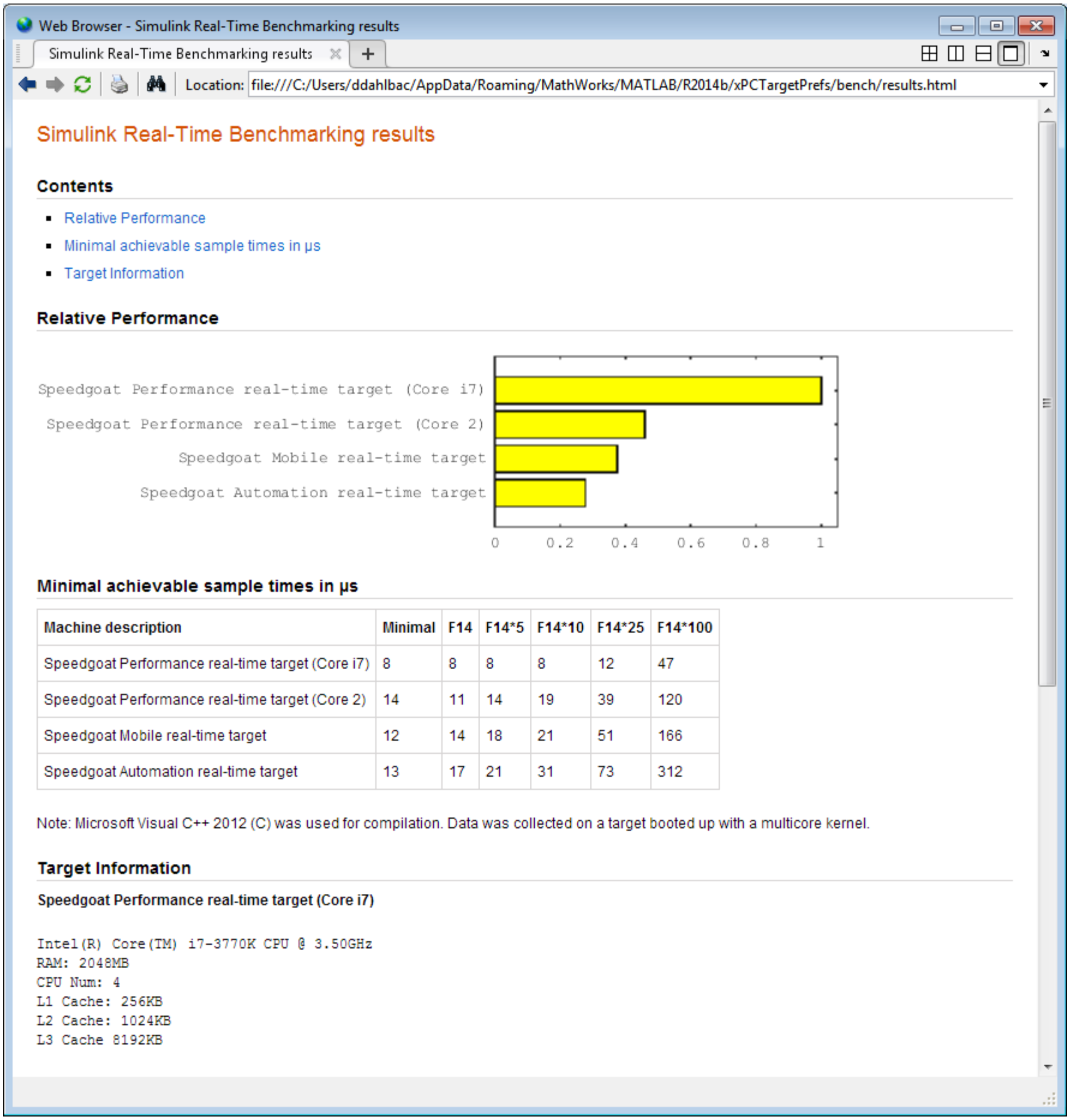
Show representative benchmark results from various target computers.

Start the target computer and run confidence test.

```
slrttest
```

Display representative results on predefined benchmarks.

```
slrtbench
```


Web Browser - Simulink Real-Time Benchmarking results
Simulink Real-Time Benchmarking results
Location: file:///C:/Users/ddahlbac/AppData/Roaming/MathWorks/MATLAB/R2014b/xPCTargetPrefs/bench/results.html
Speedgoat Performance real-time target (Core i7)
Speedgoat Performance real-time target (Core 2)
Speedgoat Mobile real-time target
Speedgoat Automation real-time target
0
0.2
0.4
0.6
0.8
1

Simulink Real-Time Benchmarking results

Contents

- Relative Performance
- Minimal achievable sample times in µs
- Target Information

Relative Performance

Minimal achievable sample times in µs

Machine description	Minimal	F14	F14*5	F14*10	F14*25	F14*100
Speedgoat Performance real-time target (Core i7)	8	8	8	8	12	47
Speedgoat Performance real-time target (Core 2)	14	11	14	19	39	120
Speedgoat Mobile real-time target	12	14	18	21	51	166
Speedgoat Automation real-time target	13	17	21	31	73	312

Note: Microsoft Visual C++ 2012 (C) was used for compilation. Data was collected on a target booted up with a multicore kernel.

Target Information

Speedgoat Performance real-time target (Core i7)

```
Intel(R) Core(TM) i7-3770K CPU @ 3.50GHz
RAM: 2048MB
CPU Num: 4
L1 Cache: 256KB
L2 Cache: 1024KB
L3 Cache 8192KB
```

slrtbench this

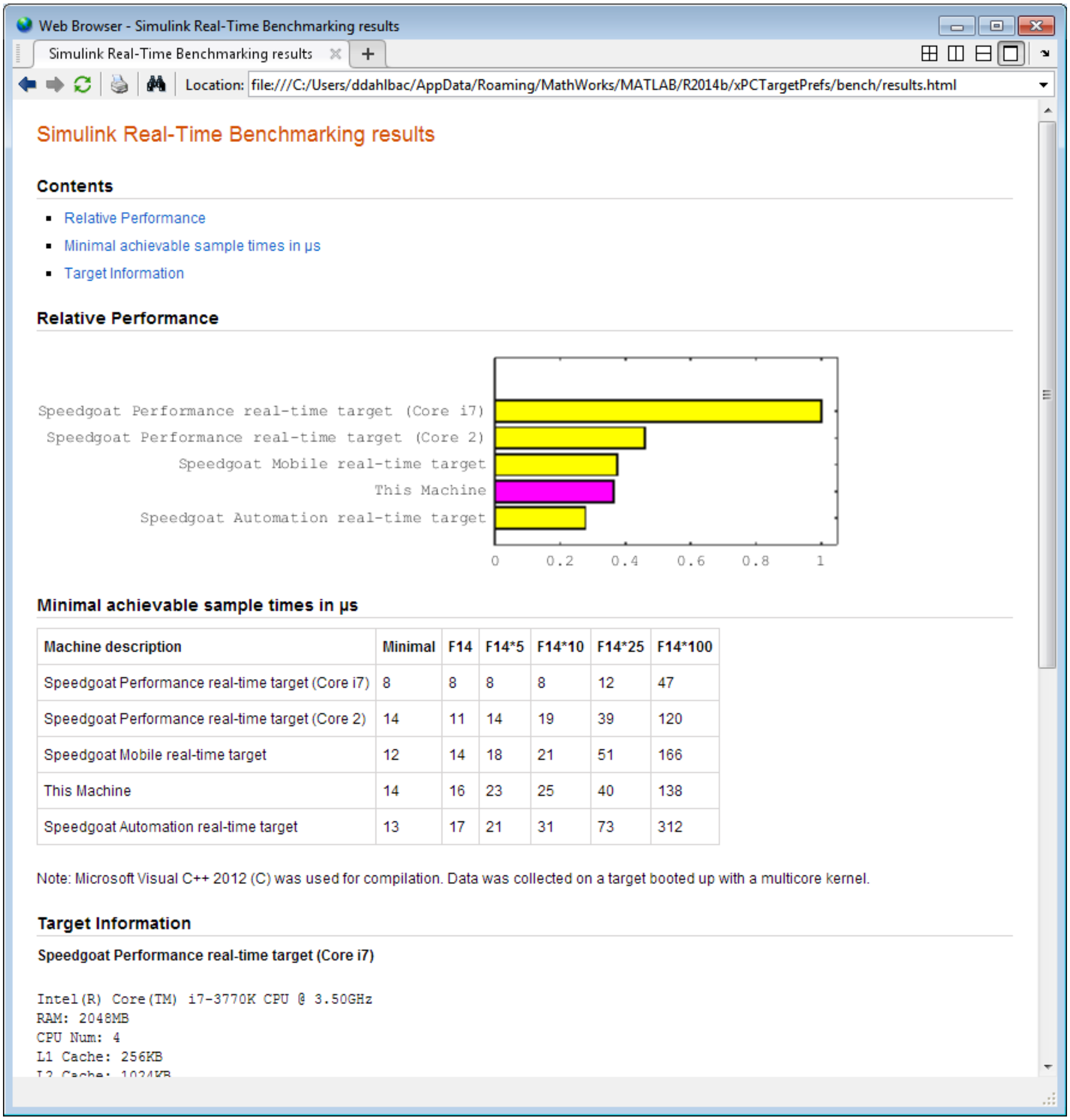
Benchmark the target computer with the predefined benchmarks.

Start the target computer and run confidence test.

```
slrttest
```

Run the benchmark models and display results.

```
slrtbench this

### Starting Simulink Real-Time build procedure
     for model: xpcminimal
### Successful completion of build procedure for model: xpcminimal
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: xpcminimal
.
.
.
### Running benchmark for model: f14tmp1
.
.
.
### Running benchmark for model: f14tmp5
.
.
.
### Running benchmark for model: f14tmp10
.
.
.
### Running benchmark for model: f14tmp25
.
.
.
### Running benchmark for model: f14tmp100
```


Web Browser - Simulink Real-Time Benchmarking results
Simulink Real-Time Benchmarking results
Location: file:///C:/Users/ddahlbac/AppData/Roaming/MathWorks/MATLAB/R2014b/xPCTargetPrefs/bench/results.html
Speedgoat Performance real-time target (Core i7)
Speedgoat Performance real-time target (Core 2)
Speedgoat Mobile real-time target
This Machine
Speedgoat Automation real-time target
0
0.2
0.4
0.6
0.8
1

Simulink Real-Time Benchmarking results

Contents

- Relative Performance
- Minimal achievable sample times in µs
- Target Information

Relative Performance

Minimal achievable sample times in µs

Machine description	Minimal	F14	F14*5	F14*10	F14*25	F14*100
Speedgoat Performance real-time target (Core i7)	8	8	8	8	12	47
Speedgoat Performance real-time target (Core 2)	14	11	14	19	39	120
Speedgoat Mobile real-time target	12	14	18	21	51	166
This Machine	14	16	23	25	40	138
Speedgoat Automation real-time target	13	17	21	31	73	312

Note: Microsoft Visual C++ 2012 (C) was used for compilation. Data was collected on a target booted up with a multicore kernel.

Target Information

Speedgoat Performance real-time target (Core i7)

```
Intel(R) Core(TM) i7-3770K CPU @ 3.50GHz
RAM: 2048MB
CPU Num: 4
L1 Cache: 256KB
L2 Cache: 1024KB
```

slrtbench this -verbose -reboot -cleanup

Benchmark the target computer with the predefined benchmarks and all control options.

Start the target computer and run confidence test.

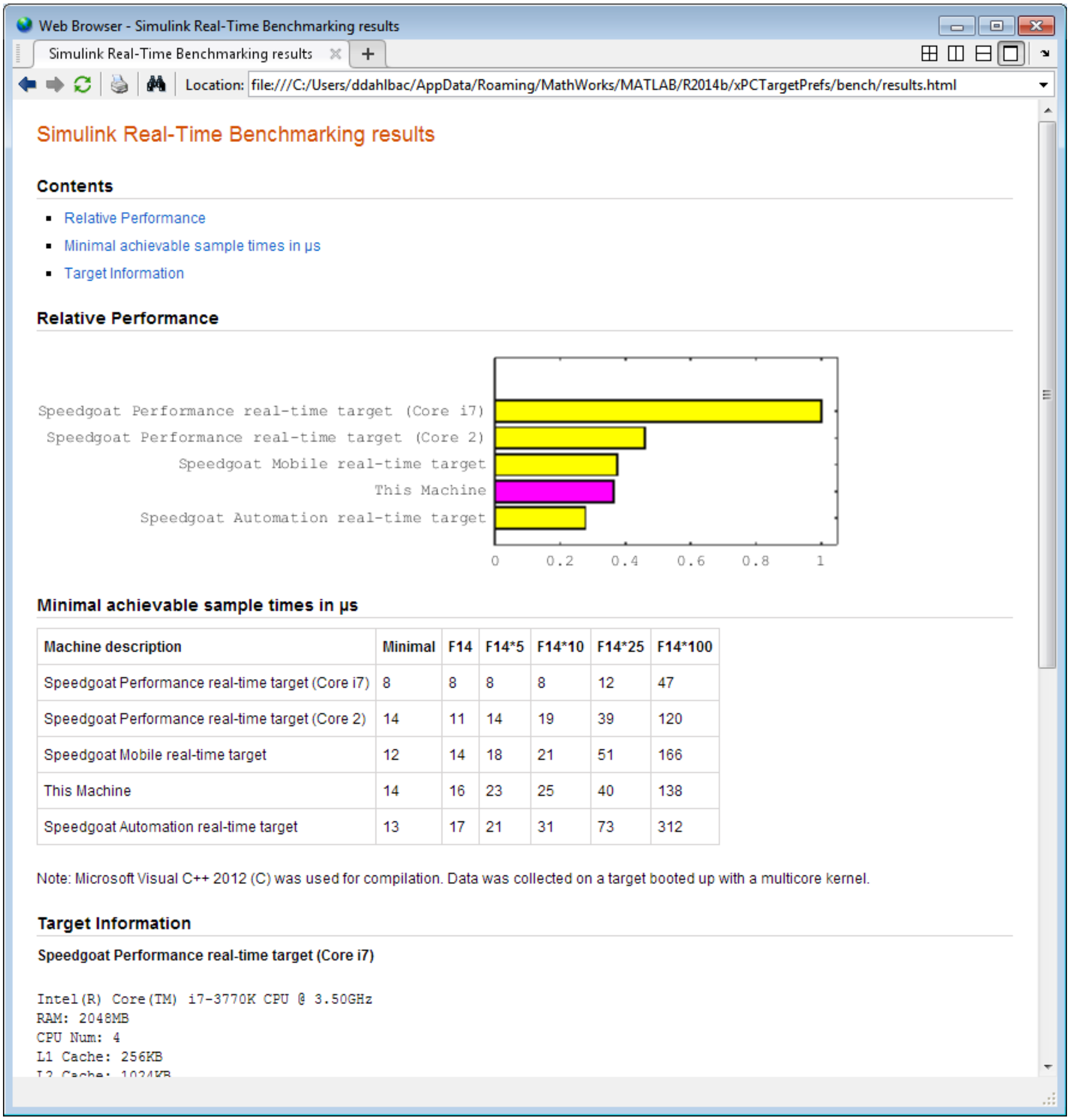
```
slrttest
```

Run the benchmark models, restart the target computer, delete build files, and display results.

```
slrtbench this -verbose -reboot -cleanup
```

```
### Starting Simulink Real-Time build procedure
     for model: xpcminimal
### Generating code into build folder: xpcminimal_xpc_rtw
### Invoking Target Language Compiler on xpcminimal.rtw
.
.
.
### Successful completion of build procedure for model:
     xpcminimal
### Looking for target: TargetPC1
### Download model onto target: TargetPC1
### Create SimulinkRealTime.target object tg
Target: TargetPC1
   Connected            = Yes
.
.
.
### Running benchmark for model: xpcminimal
### Reboot target: TargetPC1........ OK.
.
.
### Running benchmark for model: f14tmp1
### Reboot target: TargetPC1........ OK.
.
.
.
### Running benchmark for model: f14tmp5
### Reboot target: TargetPC1........ OK.
.
.
.
### Running benchmark for model: f14tmp10
### Reboot target: TargetPC1........ OK.
.
.
.
### Running benchmark for model: f14tmp25
### Reboot target: TargetPC1........ OK.
```

```
.
.
.
### Running benchmark for model: f14tmp100
### Reboot target: TargetPC1........ OK.
```


Web Browser - Simulink Real-Time Benchmarking results
Simulink Real-Time Benchmarking results
+
Location:

file:///C:/Users/ddahlbac/AppData/Roaming/MathWorks/MATLAB/R2014b/xPCTargetPrefs/bench/results.html

Simulink Real-Time Benchmarking results

Contents

- Relative Performance
- Minimal achievable sample times in µs
- Target Information

Relative Performance

Speedgoat Performance real-time target (Core i7)
Speedgoat Performance real-time target (Core 2)
Speedgoat Mobile real-time target
This Machine
Speedgoat Automation real-time target
0
0.2
0.4
0.6
0.8
1

Minimal achievable sample times in µs

Machine description	Minimal	F14	F14*5	F14*10	F14*25	F14*100
Speedgoat Performance real-time target (Core i7)	8	8	8	8	12	47
Speedgoat Performance real-time target (Core 2)	14	11	14	19	39	120
Speedgoat Mobile real-time target	12	14	18	21	51	166
This Machine	14	16	23	25	40	138
Speedgoat Automation real-time target	13	17	21	31	73	312

Note: Microsoft Visual C++ 2012 (C) was used for compilation. Data was collected on a target booted up with a multicore kernel.

Target Information

Speedgoat Performance real-time target (Core i7)

```
Intel(R) Core(TM) i7-3770K CPU @ 3.50GHz
RAM: 2048MB
CPU Num: 4
L1 Cache: 256KB
L2 Cache: 1024KB
```

slrtbench xpcosc

Use model `xpcosc` to benchmark the target computer, and then clean up build files

Start the target computer and run confidence test.

```
slrttest
```

Run benchmark on `xpcosc`, delete build files, and print results.

```
slrtbench xpcosc
```

```
### Starting Simulink Real-Time build procedure for model: xpcosc
### Successful completion of build procedure for model: xpcosc
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: xpcosc

Benchmark results for model:               xpcosc
Number of blocks in model:                 10
Elapsed time for model build (sec):        33.4
Elapsed time for model benchmark (sec):    236.7
Minimal achievable sample time (microsec): 12.4
```

slrtbench xpcosc --verbose -reboot -cleanup

Use model `xpcosc` to benchmark the target computer with all control options.

Start the target computer and run confidence test.

```
slrttest
```

Run benchmark on `xpcosc`, restart the target computer, delete build files, and print results.

```
slrtbench xpcosc -verbose -reboot -cleanup
```

```
### Starting Simulink Real-Time build procedure for model: xpcosc
### Generating code into build folder: xpcosc_slrt_rtw
### Invoking Target Language Compiler on xpcosc.rtw
.
.
.
### Successful completion of build procedure for model: xpcosc
### Looking for target: TargetPC1
### Download model onto target: TargetPC1
### Create SimulinkRealTime.target object tg
Target: TargetPC1
   Connected            = Yes
```

```
.
.
.

### Running benchmark for model: xpcosc
### Reboot target: TargetPC1........ OK

Benchmark results for model:               xpcosc
Number of blocks in model:                 10
Elapsed time for model build (sec):        29.4
Elapsed time for model benchmark (sec):    210.5
Minimal achievable sample time (microsec): 10.9
```

expected_results = slrtbench()

Return a structure array containing benchmark results showing what to expect of various target computers.

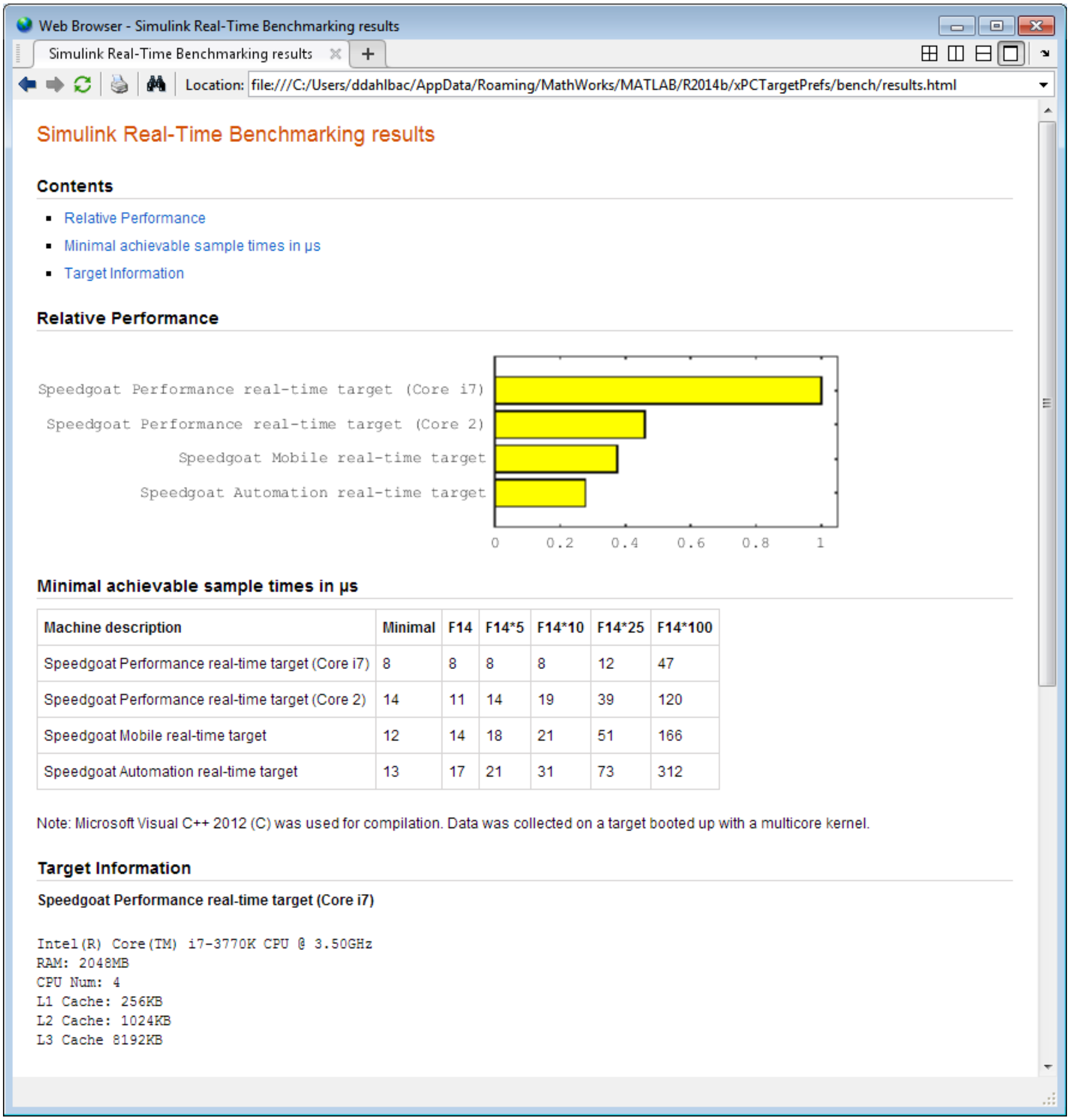
Start the target computer and run confidence test.

```
slrttest
```

Return an array with representative results for each processor type, in arbitrary order.

```
expected_results = slrtbench();
expected_results(1)
```

```
ans = 

       Machine: 'Speedgoat Performance real-time target (Core i7)'
  BenchResults: [1x6 double]
          Desc: '%  Intel(R) Core(TM) i7-3770K CPU @ 3.50GHz
% RAM: 2...'
```


Web Browser - Simulink Real-Time Benchmarking results
Simulink Real-Time Benchmarking results
Location: file:///C:/Users/ddahlbac/AppData/Roaming/MathWorks/MATLAB/R2014b/xPCTargetPrefs/bench/results.html

Simulink Real-Time Benchmarking results

Contents

- Relative Performance
- Minimal achievable sample times in μs
- Target Information

Relative Performance

Speedgoat Performance real-time target (Core i7)
Speedgoat Performance real-time target (Core 2)
Speedgoat Mobile real-time target
Speedgoat Automation real-time target
0
0.2
0.4
0.6
0.8
1

Minimal achievable sample times in μs

Machine description	Minimal	F14	F14*5	F14*10	F14*25	F14*100
Speedgoat Performance real-time target (Core i7)	8	8	8	8	12	47
Speedgoat Performance real-time target (Core 2)	14	11	14	19	39	120
Speedgoat Mobile real-time target	12	14	18	21	51	166
Speedgoat Automation real-time target	13	17	21	31	73	312

Note: Microsoft Visual C++ 2012 (C) was used for compilation. Data was collected on a target booted up with a multicore kernel.

Target Information

Speedgoat Performance real-time target (Core i7)

```
Intel(R) Core(TM) i7-3770K CPU @ 3.50GHz
RAM: 2048MB
CPU Num: 4
L1 Cache: 256KB
L2 Cache: 1024KB
L3 Cache 8192KB
```

current_results = slrtbench('xpcosc', '-verbose', '-reboot', '-cleanup')

Benchmark the target computer using the `xpcosc` model with all control options. Return a structure array with results.

Start the target computer and run confidence test.

```
slrttest
```

Build `'xpcosc'`, print build messages, run benchmark, restart the target computer, delete build files, and return results.

```
current_results = slrtbench('xpcosc','-verbose','-reboot',
     '-cleanup')

### Starting Simulink Real-Time build procedure for model: xpcosc
### Generating code into build folder: xpcosc_slrt_rtw
### Generated code for 'xpcosc' is up to date because no
     structural, parameter or code replacement library
     changes were found.
.
.
.
### Successful completion of build procedure for model: xpcosc
### Looking for target: TargetPC1
### Download model onto target: TargetPC1
### Create SimulinkRealTime.target object tg
Target: TargetPC1
   Connected            = Yes
.
.
.
### Running benchmark for model: xpcosc
### Reboot target: TargetPC1......... OK

Benchmark results for model:               xpcosc
Number of blocks in model:                 10
Elapsed time for model build (sec):        14.5
Elapsed time for model benchmark (sec):    200.5
Minimal achievable sample time (microsec): 11.9

current_results =

         Name: 'xpcosc'
      nBlocks: 10
    BuildTime: 14.4840
    BenchTime: 200.4516
```

```
        Tsmin: 1.1875e-05
```

Input Arguments

`benchmark` — Benchmark name or model name

this | *usermdl* | minimal | f14 | f14*5 | f14*10 | f14*25 | f14*100

Benchmark, specified as a literal string or string variable containing one of:

`this`	All five predefined benchmark models (`minimal`, `f14`, `f14*5`, `f14*10`, `f14*25`)
usermdl	Your model, *usermdl*.
`minimal`	Minimal model consisting of three blocks (Constant, Gain, Termination).
`f14`	Standard Simulink example `f14` (62 blocks, 10 continuous states).
`f14*5`	Five `f14` systems modeled in subsystems (310 blocks, 50 continuous states).
`f14*10`	Ten `f14` systems (620 blocks, 100 continuous states).
`f14*25`	25 `f14` systems (1550 blocks, 250 continuous states).
`f14*100`	100 `f14` systems (6200 blocks, 1000continuous states).

When using function form, enclose literal arguments in single quotes.

Example: `'this'`

Example: `'-reboot'`

Data Types: `char`

Output Arguments

`expected_results` — Results of predefined benchmarks previously run on representative target computers

struct array

Contains representative benchmark results in a structure array with element fields:

Machine	Target computer information string containing CPU type, CPU speed, compiler
BenchResults	Target computer benchmark performance for all five predefined benchmarks
Desc	Target computer descriptor string containing machine type, RAM size, cache size

`current_results` — Current results of specified benchmark
struct

Contains actual benchmark results in a structure with fields:

Name	Benchmark name
nBlocks	Number of blocks in benchmark
BuildTime	Elapsed time in seconds to build benchmark
BenchTime	Elapsed time in seconds to run benchmark
Tsmin	Minimal achievable sample time in seconds for benchmark

More About

Tips

- Before you run `slrtbench`, you must be able to start the target computer, connect the development computer to the target computer, and run the confidence test, `slrttest`, with no failures.
- After running `slrtbench` on your model and system, set your model sample time to the minimal achievable sample time value reported. Smaller sample times overload the target computer.
- The stored benchmark results were collected with **Multicore CPU support** disabled. When evaluating your system, temporarily disable this target setting using `slrtexplr`.

- The stored benchmark models were compiled using a sampling of the supported compilers. When evaluating your system, find the closest match to the compiler that you are using.
- Benchmark `minimal` has neither continuous nor discrete states. It provides an indication of the target computer interrupt latencies.
- http://www.mathworks.com/support/compilers/current_release/

See Also

`slrttest`

Introduced in R2014a

slrtdrivertool

Construct skeleton for custom driver

Syntax

```
slrtdrivertool
```

Description

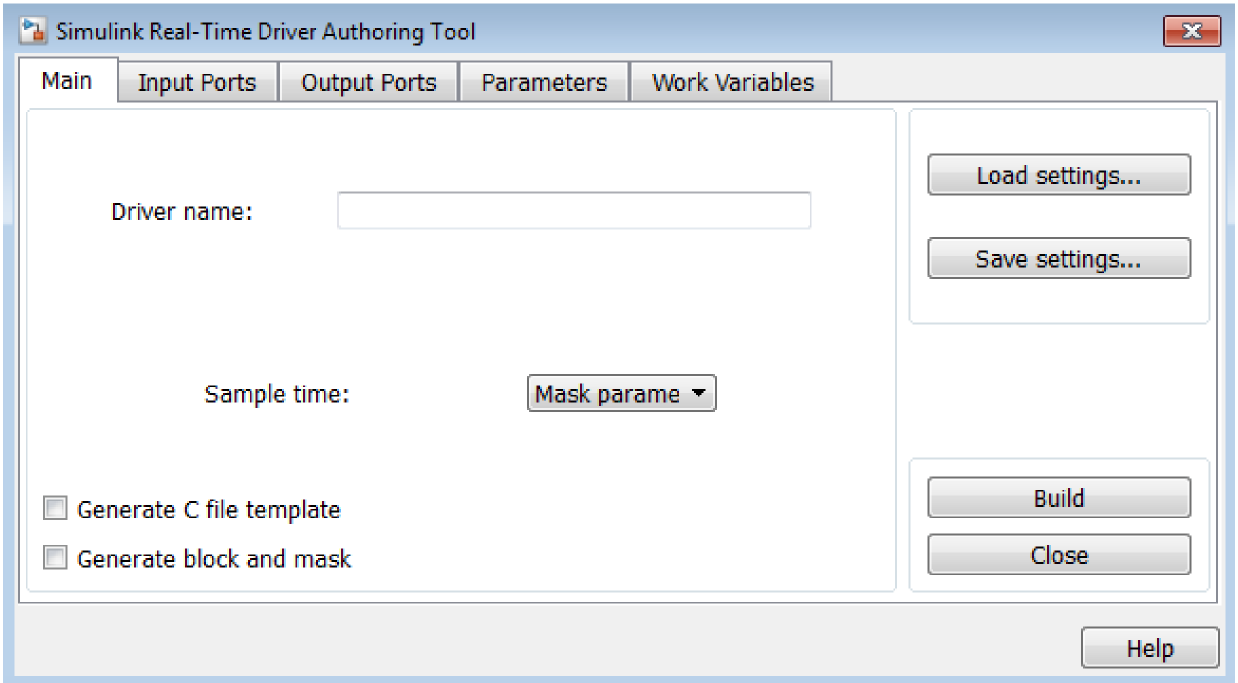
`slrtdrivertool` opens the Simulink Real-Time Driver Authoring Tool. Using this tool, you can:

- Define the driver name.
- Specify how the sample time is defined (inherited or as a mask parameter).
- Define input and output ports.
- Define parameters and working variables.
- Generate a C file template (optional).
- Generate a block and mask dialog box (optional).
- Save and load settings.
- Build a skeleton driver.

Examples

Define a skeleton driver

```
slrtdrivertool
```



Driver name:

Sample time: Mask parame

□ Generate C file template

□ Generate block and mask

Introduced in R2014a

slrtexplr

Configure target computer and real-time application for execution

Syntax

```
slrtexplr
```

Description

Typing `slrtexplr` at the MATLAB command prompt opens Simulink Real-Time Explorer.

From within Simulink Real-Time Explorer, you can export a session as a standalone executable that runs without MATLAB.

When you run Simulink Real-Time Explorer from within MATLAB, you have available the full capabilities of Simulink Real-Time Explorer. When you run it as a standalone executable, you have available a subset of the capabilities of Simulink Real-Time Explorer.

- Environment configuration
  - Configure and view communication parameters.
  - Configure target computer settings
  - Configure target computer startup
  - Browse target computer file system.
- Control
  - Load, run, and unload real-time applications on the target computer.
  - Connect to and disconnect from the target computer.
  - Change stop time and sample times without regenerating code.
  - Record task execution time during or after last run.
- Instrumentation
  - Create graphical instrument panels for acquiring signals and tuning parameters.

  - Save and load instrument panels.
  - Start and stop instrument panels.
  - Use instrument panels to interact with application.
- Signal acquisition
  - Create, save, and load signal groups.
  - Monitor signals.
  - Add and configure host, target, or file scopes.
  - Attach signals to or remove signals from scopes.
  - Start and stop scopes.
  - Attach signals to instruments.
- Parameter tuning
  - Create, save, and load parameter groups.
  - Display and tune parameters.
  - Attach parameters to instruments.
- Window configuration
  - Make multiple workspaces visible simultaneously.
  - Move workspaces around the window.
  - Export model configuration as a standalone executable.
  - Save and restore model configuration layouts.

When you run Simulink Real-Time Explorer as a standalone executable, it has the following restrictions:

- You cannot change the communication parameters that the interface uses to communicate with the target computer. Before you export the Simulink Real-Time Explorer configuration, configure and test the communication parameters.
- For each instrument, the exporting software records the real-time application and target computer environment with which it is associated. If you rename the target computer, update the **TargetName** parameter for each instrument to maintain the connection to the real-time application.
- You cannot load or unload a real-time application from the standalone executable. Before you start the executable, start the real-time application on the target computer.

- You can access only instrument panels and windows that you loaded before you exported the configuration.
- You cannot access the model hierarchy from the standalone executable.
- You can access only signals in signal groups that you loaded before you exported the configuration.

  You cannot move a signal from one signal group to another or create or load a new signal group.
- You can access only parameters in parameter groups that you loaded before you exported the configuration.

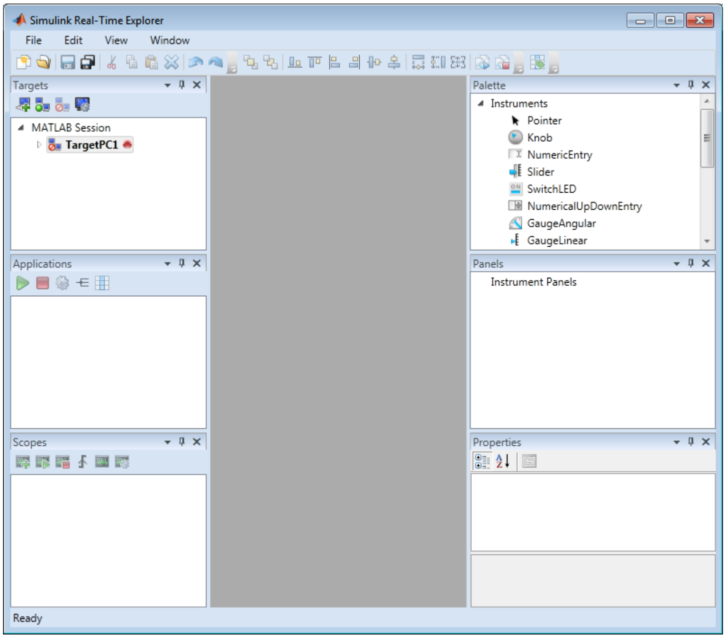
  You cannot move a parameter from one parameter group to another or create or load a new parameter group.
- You cannot save session layouts. If you close a window, you can restore the original layout using **File** > **Restore Original View**.

Examples

Default

Open Simulink Real-Time Explorer

```
slrtexplr
```



- “Ethernet Link Setup”
- “Serial Link Setup”
- “Target Computer Settings”
- “Target Boot Methods”
- “Execute Real-Time Application Using Simulink Real-Time Explorer”
- “Monitor Signals Using Simulink Real-Time Explorer”
- “Create Target Scopes Using Simulink Real-Time Explorer”
- “Create Host Scopes Using Simulink Real-Time Explorer”
- “Create File Scopes Using Simulink Real-Time Explorer”
- “Tune Parameters Using Simulink Real-Time Explorer”
- “Simulink Real-Time Explorer Run Within MATLAB”
- “Simulink Real-Time Explorer Deployed as Standalone Executable”

More About

- “Instruments and Instrument Panels”

Introduced in R2014a

slrtgetCC

Compiler settings for development computer environment

Syntax

```
slrtgetCC
type = slrtgetCC
type = slrtgetCC('Type')
location = slrtgetCC('Location')
[type,location] = slrtgetCC
slrtgetCC('supported')
slrtgetCC('installed')
[compilers] = slrtgetCC('installed')
```

Description

`slrtgetCC` displays the compiler type and location in the Command Window.

`type = slrtgetCC` and `type = slrtgetCC('Type')` both return the compiler type in `type`.

`location = slrtgetCC('Location')` returns the compiler location in `location`.

The `mex -setup` command sets the default compiler for Simulink Real-Time builds, provided the MEX compiler is a supported Microsoft compiler. `slrtgetCC` returns the result of the `slrtsetCC` command only, not the result of the `mex` command. If `slrtgetCC` returns an empty string as `location`, Simulink Real-Time is using the MEX compiler.

`[type,location] = slrtgetCC` returns the compiler type and its location in `type` and `location`.

`slrtgetCC('supported')` displays the compiler versions supported by the Simulink Real-Time environment.

`slrtgetCC('installed')` displays the supported compilers installed on the development computer.

`[compilers] = slrtgetCC('installed')` returns in a structure the supported compilers installed on the development computer.

Examples

Display compiler type and location

```
slrtgetCC
```

```
Compiler Settings:

 Type = VisualC
 Location = C:\Program Files (x86)\Microsoft Visual Studio 10.0
```

Return compiler type

```
type = slrtgetCC('Type')
```

```
type =

VisualC
```

Return compiler location

```
location = slrtgetCC('Location')
```

```
location =

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

Return compiler type and location

```
[type, location] = slrtgetCC
```

```
type =

VisualC


location =

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

Display supported compilers

```
slrtgetCC('supported')
```

```
List of C++ Compilers supported by Simulink Real-Time:

Name                                              Version   Service
                                                            Packs
Microsoft Visual C++ Compilers 2008               9.0       1
Microsoft Visual C++ Compilers 2010               10.0      1
Microsoft Visual C++ Compilers 2012               11.0
Microsoft Visual C++ Compilers (Windows SDK) 2010 10.0      1
```

Display supported compilers installed

```
slrtgetCC('installed')

List of installed C++ Compilers:

Name: Microsoft Visual C++ Compilers 2008 Professional Edition
      (SP1)
Location: c:\Program Files (x86)\Microsoft Visual Studio 9.0

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

Return supported compilers installed

```
[compilers] = slrtgetCC('installed')
compilers(1)

compilers =

1x2 struct array with fields:

    Type
    Name
    Location

ans =

        Type: 'VisualC'
        Name: 'Microsoft Visual C++ Compilers 2008 Professional
               Edition (SP1)'
    Location: 'c:\Program Files (x86)\Microsoft Visual Studio 9.0'
```

Output Arguments

`type` — Type of compiler
`VisualC`

Simulink Real-Time supports the Microsoft Visual Studio C compiler only.

`location` — Folder path to compiler on development computer
string

`compilers` — Array of structures containing compiler type, name, and location
array of structures

More About

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

See Also

`mex` | `slrtsetCC`

Introduced in R2014a

slrtpingtarget

Test communication between development and target computers

Syntax

```
slrtpingtarget
```

```
slrtpingtarget target_computer_name
```

Description

Returns `success` if the Simulink Real-Time kernel is loaded and running, and communication is working between the development and target computers. Otherwise, returns `failed`.

`slrtpingtarget` without an argument returns `success` if the development computer and the default target computer can communicate using the settings for that target computer. Otherwise, returns `failed`.

`slrtpingtarget target_computer_name` returns `success` if the development computer can communicate with target computer `target_computer_name` using the settings for that target computer. Otherwise, returns `failed`.

Examples

Check communication with default target computer

```
slrtpingtarget
```

Check communication with specified target computer

```
slrtpingtarget TargetPC1
```

Input Arguments

`target_computer_name` — Name of specific target computer

`TargetPC1` | `TargetPC2` | ...

Name property of a particular target computer environment object. The default name is `TargetPC1`.

When using function form, enclose the argument in single quotes (`'TargetPC1'`).

Example: `TargetPC1`

Data Types: `char`

Introduced in R2014a

slrtsetCC

Compiler settings for development computer environment

Syntax

```
slrtsetCC setup
slrtsetCC 'type' 'location'
```

Description

`slrtsetCC setup` queries the development computer for installed C compilers supported by the Simulink Real-Time environment. You can then select the C compiler.

The command `mex -setup` sets the default compiler for Simulink Real-Time builds, provided the MEX compiler is a supported Microsoft compiler. Use `slrtsetCC('setup')` only if you must specify different compilers for MEX and Simulink Real-Time.

`slrtsetCC 'type' 'location'` sets the compiler type and location.

To return to the default MEX compiler from a setting by `slrtsetCC`, type `slrtsetCC 'VisualC' ''`, setting the compiler location to the empty string.

Examples

Compiler selection

```
slrtsetCC setup
```

```
Select your compiler for Simulink Real-Time.

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

```
[0] None


Compiler:2

Verify your selection:

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

Are these correct [y]/n?y

Done...
```

Compiler specification

```
slrtsetCC 'VisualC',
     'C:\Program Files (x86)\Microsoft Visual Studio 10.0'
```

Input Arguments

`type` — Type of compiler

`VisualC` (default)

`type` must be `VisualC`, representing the Microsoft Visual Studio C compiler.

Example: `'VisualC'`

Data Types: `char`

`location` — Folder path to compiler on development computer

string

Data Types: `char`

More About

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

See Also

`mex` | `slrtgetCC`

Introduced in R2014a

slrttest

Test Simulink Real-Time installation

Syntax

```
slrttest
slrttest noreboot
slrttest target_name,___
```

Description

`slrttest` is a confidence test that checks the following tasks:

- Initiate communication between the development and target computers.
- Restart the target computer to reset the target environment.
- Build a real-time application on the development computer.
- Download a real-time application to the target computer.
- Check communication between the development and target computers using commands.
- Execute a real-time application.
- Compare the results of a simulation and the real-time application run.

`slrttest noreboot` skips the restart test on the default target computer. Use this option if the target hardware does not support software restart.

`slrttest target_name,___` runs the tests on the target computer identified by `target_name`.

Examples

Test default target computer

Target computer must be running and physically connected to the development computer.

```
slrttest
```

Test default target computer, skipping reboot test

Target computer must be running and physically connected to the development computer.

```
slrttest noreboot
```

Test specified target computer, skipping reboot test

Target computer must be running and physically connected to the development computer.

```
slrttest 'TargetPC1' noreboot
```

Input Arguments

`target_name` — Specifies target name

string

The target name string is case sensitive.

Example: `'TargetPC1'`

More About

- “Troubleshooting in Simulink Real-Time”

Introduced in R2014a

SimulinkRealTime.addTarget

Add new Simulink Real-Time target object

Syntax

```
SimulinkRealTime.addTarget('target_name')
```

Description

`SimulinkRealTime.addTarget('target_name')` adds the definition for a new target computer, represented by the name `'mytarget'`. It returns an object of type `SimulinkRealTime.targetSettings` corresponding to the new target computer.

Examples

Add a new Simulink Real-Time target object `'TargetPC2'` to the system:

```
tg = SimulinkRealTime.addTarget('TargetPC2')
```

The `tg` variable contains the attributes of the new target computer.

See Also

`SimulinkRealTime.getTargetSettings` | `SimulinkRealTime.removeTarget`

Introduced in R2014a

SimulinkRealTime.copyFileToHost

Copy file from target computer to development computer

Syntax

```
SimulinkRealTime.copyFileToHost(file_name)
SimulinkRealTime.copyFileToHost(target_obj,file_name)
```

Description

`SimulinkRealTime.copyFileToHost(file_name)` copies file `file_name` from the default target computer to the development computer.

`SimulinkRealTime.copyFileToHost(target_obj,file_name)` copies file `file_name` from the target computer represented by `target_obj` to the development computer.

Examples

Copy File by Name from Default Target Computer

Copy file from current folder on default target computer.

```
SimulinkRealTime.copyFileToHost('data.dat')
```

Copy File by Full Path from Specified Target Computer

Copy file from full path location on target computer `TargetPC1`.

```
tg = slrt('TargetPC1');
SimulinkRealTime.copyFileToHost(tg,'c:\xpcosc\data1.dat')
```

Input Arguments

`target_obj` — Name of a target computer or a variable containing a target computer object

`string | object`

If the argument is a string, it must be the name assigned to a previously configured target computer.

If the argument is a variable containing an object, it must be a `SimulinkRealTime.target` object representing a previously configured target computer.

Example: ‘`TargetPC1`’

Example: `tg`

Data Types: `char` | `struct`

`file_name` — Name of a file on the target computer

`file name string` | `full path name string`

If the argument is a file name, the file must be in the current folder on the target computer, as indicated by the function `SimulinkFileSystem.pwd`.

The file is transferred from the target and written with the same file name to the current folder on the development computer.

Example: `'myFile.txt'`

Example: `'c:\subDir\myFile.txt'`

Data Types: `char`

See Also

`SimulinkRealTime.copyFileToTarget` | `SimulinkRealTime.fileSystem.cd` | `SimulinkRealTime.fileSystem.dir` | `SimulinkRealTime.fileSystem.pwd`

Introduced in R2014a

SimulinkRealTime.copyFileToTarget

Copy file from development computer to target computer

Syntax

```
SimulinkRealTime.copyFileToTarget(file_name)
SimulinkRealTime.copyFileToTarget(target_obj,file_name)
```

Description

`SimulinkRealTime.copyFileToTarget(file_name)` copies file `file_name` from the development computer to the default target computer.

`SimulinkRealTime.copyFileToTarget(target_obj,file_name)` copies file `file_name` from the development computer to the target computer represented by `target_obj`.

Examples

Copy File to Default Target Computer Top Folder

Copy file from current folder on development computer to top folder on default target computer.

```
SimulinkRealTime.copyFileToTarget('data.dat')
```

Copy File to Specified Target Computer by Full Path

Copy file from current folder on development computer to full path location on target computer `TargetPC1`.

```
tg = slrt('TargetPC1');
```

```
SimulinkRealTime.copyFileToTarget(tg,'c:\xpcosc\data1.dat')
```

Input Arguments

`target_obj` — Name of a target computer or a variable containing a target computer object
`string` | `object`

If the argument is a string, the string must contain the name assigned to a previously configured target computer.

If the argument is a variable containing an object, the object must be a `SimulinkRealTime.target` object representing a previously configured target computer.

Example: ‘`TargetPC1`’

Example: `tg`

Data Types: `char` | `struct`

`file_name` — Name of a file in the current folder on the development computer
`file name string` | `full path name string`

The file being copied must exist in the current folder on the development computer.

If the argument is a file name, the file is copied to the current folder on the target computer, as indicated by the function `SimulinkFileSystem.pwd`.

If the argument is a path name, the file portion of the path name is extracted as the development computer file name. The file is copied to the location indicated by the path name. The folder must exist on the target computer.

Example: `'myFile.txt'`

Example: `'c:\subDir\myFile.txt'`

Data Types: `char`

See Also

`SimulinkRealTime.copyFileToHost` | `SimulinkRealTime.fileSystem.cd` | `SimulinkRealTime.fileSystem.dir` | `SimulinkRealTime.fileSystem.pwd`

Introduced in R2014a

SimulinkRealTime.createBootImage

Create Simulink Real-Time boot disk or DOS Loader files

Syntax

```
SimulinkRealTime.createBootImage
SimulinkRealTime.createBootImage(target_name)
SimulinkRealTime.createBootImage(target_object)
```

Description

`SimulinkRealTime.createBootImage` creates a boot image for the default target computer in the form of a boot floppy disk, a boot CD or DVD, a network boot image, or DOS Loader kernel image files.

`SimulinkRealTime.createBootImage(target_name)`creates a boot image for the target computer indicated by the `target_name` string.

`SimulinkRealTime.createBootImage(target_object)`creates a boot image for the target computer indicated by `target_object`, a variable containing a target object.

The form of the boot image depends upon the value of the `TargetBoot` environment property.

- `BootFloppy` — To create a boot floppy disk, the software prompts you to insert an empty formatted disk into the drive. The software writes the kernel image onto the disk and displays a summary of the creation process.
- `CDBoot` — To create a CD or DVD boot disk, the software prompts you to insert an empty formatted CD or DVD into the drive. The software writes the kernel image onto the CD or DVD and displays a summary of the creation process.
- `NetworkBoot` — To create a network boot image, the software starts the network boot server process.
- `DOSLoader` — To create DOS Loader files, the software writes kernel image and DOS Loader files into a designated location on the development computer. You can then copy the files to the target computer hard drive, to a floppy disk, or to a flash drive.

- StandAlone — To create files for a standalone application, you must separately compile and download a combined kernel and real-time application. SimulinkRealTime.createBootImage does not generate a standalone application.

To update the TargetBoot environment property:

```
tg = SimulinkRealTime.getTargetSettings
tg.TargetBoot = new_value
```

If you update the environment, you must update the boot image with the function SimulinkRealTime.createBootImage.

Examples

To create a boot image for the default target computer, in the Command Window, type:

```
SimulinkRealTime.createBootImage
```

To create a boot image for the target computer TargetPC1, type:

```
SimulinkRealTime.createBootImage('TargetPC1')
```

To create a boot image for target computer object target_object, type:

```
target_object = SimulinkRealTime.addTarget('TargetPC2');
SimulinkRealTime.createBootImage(target_object)
```

See Also

SimulinkRealTime.getTargetSettings

Introduced in R2014a

SimulinkRealTime.getSupportInfo

Diagnostic information to troubleshoot configuration issues

Syntax

```
SimulinkRealTime.getSupportInfo
SimulinkRealTime.getSupportInfo('-a')
```

Arguments

`'-a'`	Appends diagnostic information to an existing `slrtinfo.txt` file. If this file does not exist, this function creates the file in the current folder. Enter the argument as a string.

Description

`SimulinkRealTime.getSupportInfo` returns diagnostic information for troubleshooting Simulink Real-Time configuration issues. This function generates and saves the information in the `slrtinfo.txt` file, in the current folder. If the file `slrtinfo.txt` already exists, this function overwrites it with the new information.

`SimulinkRealTime.getSupportInfo('-a')` appends the diagnostic information to the `slrtinfo.txt` file, in the current folder. If the file `slrtinfo.txt` does not exist, this function creates it.

You can send the file `slrtinfo.txt` to MathWorks® support for evaluation and guidance. To create this file, you must have write permission for the current folder.

Caution The file `slrtinfo.txt` can contain information sensitive to your organization. Before sending this file to MathWorks, review the contents.

Introduced in R2014a

SimulinkRealTime.getTargetSettings

Get target computer environment settings

Syntax

```
SimulinkRealTime.getTargetSettings
SimulinkRealTime.getTargetSettings(target_computer_name)
settings_object = SimulinkRealTime.getTargetSettings(___)

SimulinkRealTime.getTargetSettings('-all')
settings_object_vector = SimulinkRealTime.getTargetSettings('-all')
```

Description

`SimulinkRealTime.getTargetSettings` displays the environment settings for the default computer.

`SimulinkRealTime.getTargetSettings(target_computer_name)` displays the environment settings for a particular target computer.

`settings_object = SimulinkRealTime.getTargetSettings(___)` returns an environment object representing a target computer.

`SimulinkRealTime.getTargetSettings('-all')` displays a list of environment objects representing all defined target computers.

`settings_object_vector = SimulinkRealTime.getTargetSettings('-all')` returns a vector of environment objects representing all target computers.

Examples

Display Settings for Default Target

Display environment settings for default target computer.

```
SimulinkRealTime.getTargetSettings
```

```
Simulink Real-Time Target Settings

    Name                      : TargetPC1

    TargetRAMSizeMB           : Auto
    MaxModelSize              : 1MB
    SecondaryIDE              : off
    NonPentiumSupport         : off
    MulticoreSupport          : on
    LegacyMultiCoreConfig     : off
    USBSupport                : on
    ShowHardware              : off
    EthernetIndex             : 0

    HostTargetComm            : TcpIp
    TcpIpTargetAddress        : 10.10.10.15
    TcpIpTargetPort           : 22222
    TcpIpSubNetMask           : 255.255.255.0
    TcpIpGateway              : 10.10.10.100
    RS232HostPort             : COM1
    RS232Baudrate             : 115200
    TcpIpTargetDriver         : Auto
    TcpIpTargetBusType        : PCI
    TcpIpTargetISAMemPort     : 0x300
    TcpIpTargetISAIRQ         : 5

    TargetScope               : Enabled

    TargetBoot                : NetworkBoot
    TargetMACAddress          : 90:e2:ba:17:5d:15
```

Display Settings for Specific Target

Display environment settings for a specific target computer.

```
SimulinkRealTime.getTargetSettings('TargetPC2')
```

```
Simulink Real-Time Target Settings

    Name                      : TargetPC2

    TargetRAMSizeMB           : Auto
    MaxModelSize              : 1MB
    SecondaryIDE              : off
    NonPentiumSupport         : off
    MulticoreSupport         : off
    LegacyMultiCoreConfig    : off
    USBSupport               : on
    ShowHardware             : off
    EthernetIndex            : 0

    HostTargetComm           : RS232
    TcpIpTargetAddress       :
    TcpIpTargetPort          : 22222
    TcpIpSubNetMask          : 255.255.255.0
    TcpIpGateway             : 255.255.255.255
    RS232HostPort            : COM1
    RS232Baudrate            : 115200
    TcpIpTargetDriver        : Auto
    TcpIpTargetBusType       : PCI
    TcpIpTargetISAMemPort    : 0x300
    TcpIpTargetISAIRQ        : 5

    TargetScope              : Enabled

    TargetBoot               : BootFloppy
    BootFloppyLocation       :
```

Display Settings for All Targets

Display environment settings for all target computers.

```
SimulinkRealTime.getTargetSettings('-all')
```

```
 NumTargets: 2
    Targets   : Name                Communication Settings...
                TargetPC1 (Default) TcpIp:10.10.10.15:22222...
                TargetPC2           RS232:COM1:115200(BPS)...

Simulink Real-Time Target Settings

    Name                     : TargetPC1
.
.
.
    HostTargetComm           : TcpIp
.
.
.
    TargetBoot               : NetworkBoot
    TargetMACAddress             : 00:01:29:55:3c:bb


Simulink Real-Time Target Settings

    Name                         : TargetPC2
.
.
.
    HostTargetComm               : RS232
.
.
.
    TargetBoot                   : BootFloppy
    BootFloppyLocation           :
```

Access Settings for Specific Target

Retrieve an environment settings object for a specific target computer. Use it to access a setting.

```
settings_object = SimulinkRealTime.getTargetSettings('TargetPC1');
settings_object.HostTargetComm
```

```
ans =

TcpIp
```

Access Settings for Multiple Targets

Loop through vector of environment settings objects. Print name and communication mode settings.

```
sov = SimulinkRealTime.getTargetSettings('-all');
ii = 1;
while ii <= length(sov)
   disp(sprintf('%s HostTargetComm is %s',
        sov(ii).Name, sov(ii).HostTargetComm))
ii = ii + 1;
end
```

```
TargetPC1 HostTargetComm is TcpIp
```

```
TargetPC2 HostTargetComm is RS232
```

Input Arguments

`target_computer_name` — Name of target computer

string

The name-string of a target computer.

Example: `'TargetPC1'`

Data Types: `char`

Output Arguments

`settings_object` — Settings object representing target computer

object

Object containing target computer environment settings.

Data Types: `struct`

`settings_object_vector` — Vector of settings objects representing target computers

vector

Vector of objects containing target computer environment settings representing one or more target computers

Data Types: `struct`

Introduced in R2014a

SimulinkRealTime.pingTarget

Test communication between development and target computers

Syntax

```
SimulinkRealTime.pingTarget
```

```
SimulinkRealTime.pingTarget(target_computer_name)
```

Description

Returns `success` if the Simulink Real-Time kernel is loaded and running, and communication is working between the development and target computers. Otherwise, returns `failed`.

`SimulinkRealTime.pingTarget` without an argument returns `success` if the development computer and the default target computer can communicate using the settings for the default computer. Otherwise, returns `failed`.

`SimulinkRealTime.pingTarget(target_computer_name)` returns `success` if the development computer can communicate with target computer `target_computer_name` using the settings for target computer `target_computer_name`. Otherwise, returns `failed`.

Enclose the argument in single quotes (`'TargetPC1'`).

Examples

Check communication with default target computer

```
SimulinkRealTime.pingTarget
```

Check communication with specified target computer

```
SimulinkRealTime.pingTarget('TargetPC1')
```

Input Arguments

`target_computer_name` — Name of specific target computer

`'TargetPC1'` | `'TargetPC2'` | ...

Name property of a particular target computer environment object. The default name is `'TargetPC1'`.

Example: `TargetPC1`

Data Types: `char`

Introduced in R2014a

SimulinkRealTime.removeTarget

Remove environment data associated with target name

Syntax

```
SimulinkRealTime.removeTarget('target_name')
```

Description

`SimulinkRealTime.removeTarget('target_name')` removes the definitions and settings for the target computer represented by `'target_name'` from the system. The target objects associated with that target become invalid. If you remove the environment data for the default target computer, the next target object becomes the default target computer. Do not remove the environment data for the last target computer.

Examples

Remove the environment data for `'TargetPC2'` from the system:

```
SimulinkRealTime.removeTarget('TargetPC2')
```

See Also

`SimulinkRealTime.addTarget` | `SimulinkRealTime.getTargetSettings`

Introduced in R2014a

SimulinkRealTime.utils.bytes2file

Generate file for use by real-time From File block

Syntax

```
SimulinkRealTime.utils.bytes2file(filename,var1,. . .,varn)
```

Arguments

`filename`	Name of the data file from which the real-time From File block distributes data.
`var1,. . .,varn`	Column of data to be output to the model.

Description

`SimulinkRealTime.utils.bytes2file(filename,var1,. . .,varn)` outputs one column of `var1, . . .,varn` from file `filename` at every time step. All variables must have the same number of columns. The number of rows and the data types can be different.

Note: If the data is organized so that a row, not a column, refers to a time step, pass the transpose of the variable to `SimulinkRealTime.utils.bytes2file`. To optimize file writes, organize the data in columns.

Examples

To use the real-time From File block to output a variable `errorval` (single precision, scalar) and `velocity` (double, width 3) at every time step, you can generate the file with the command:

```
SimulinkRealTime.utils.bytes2file('myfile', errorval, velocity)
```

errorval has class 'single' and dimensions [1 x N] and velocity has class 'double' and dimensions [3 x N].

Set up the real-time From File block to output the following number of bytes at every sample time:

```
28 bytes
(1 * sizeof('single') + 3 * sizeof('double'))
```

Introduced in R2014a

SimulinkRealTime.utils.createInstrumentationModel

Construct skeleton for user interface model

Syntax

```
SimulinkRealTime.utils.createInstrumentationModel(system_name)
```

Description

`SimulinkRealTime.utils.createInstrumentationModel(system_name)` generates a skeleton Simulink instrumentation model containing `To Target` and `From Target` blocks. The model is based on tagged block parameters and tagged signals defined in the Simulink Real-Time model used to build the real-time application.

Examples

Generate an interface model

```
SimulinkRealTime.utils.createInstrumentationModel('xpcosc')
```

Input Arguments

`system_name` — Name of system for which to create an interface model

`'xpcosc'`

Model must contain tagged signals or block parameters.

Data Types: `char`

Introduced in R2014a

SimulinkRealTime.utils.getFileScopeData

Read real-time Scope file format data

Syntax

```
matlab_data = SimulinkRealTime.utils.getFileScopeData(slrtfile_name)
matlab_data = SimulinkRealTime.utils.getFileScopeData(slrtfile_data)
```

Description

`matlab_data = SimulinkRealTime.utils.getFileScopeData(slrtfile_name)` takes as an argument the name of a development computer file containing a vector of byte data (`uint8`). Before using this function, copy the file from the target computer using the `SimulinkRealTime.copyFileToHost` method.

`matlab_data = SimulinkRealTime.utils.getFileScopeData(slrtfile_data)` takes as an argument a MATLAB variable containing a vector of byte data (`uint8`). Before using this function, load the data into memory from a file on the target file system using the `SimulinkRealTime.fileSystem.fread` method.

Examples

Using `slrtfile_name` argument to read file and plot results

Upload file `'data.dat'` to the host. Read the file on the host. Plot the results.

Upload file `'data.dat'` from the target computer to the development computer.

```
SimulinkRealTime.copyFileToHost('data.dat')
```

Read the file and process its data into MATLAB format.

```
matlab_data = SimulinkRealTime.utils.getFileScopeData('data.dat');
```

Plot the signal data (column 1) on the `Y` axis against time (column 2) on the `X` axis.

```
plot(matlab_data.data(:,2), matlab_data.data(:,1))
xlabel(matlab_data.signalNames(2))
ylabel(matlab_data.signalNames(1))
```

Using `slrtfile_data` argument to store data, convert data to MATLAB format, and plot results

Read file `'data.dat'` on the target computer from the host. Store the data in a MATLAB workspace variable. Convert the data to MATLAB format. Plot the results.

Read file `'data.dat'` from the development computer using file system commands.

```
fs = SimulinkRealTime.fileSystem;
h = fopen(fs, 'data.dat');
slrtfile_data = fread(fs, h);
fclose(fs,h)
```

Process data from the workspace variable into MATLAB format.

```
matlab_data =
     SimulinkRealTime.utils.getFileScopeData(slrtfile_data);
```

Plot the signal data (column 1) on the `Y` axis against time (column 2) on the `X` axis.

```
plot(matlab_data.data(:,2), matlab_data.data(:,1))
xlabel(matlab_data.signalNames(2))
ylabel(matlab_data.signalNames(1))
```

Input Arguments

`slrtfile_name` — Name of file from which to read real-time `Scope` file format data

`'data.dat'`

File must contain a vector of `uint8` data.

Data Types: `char`

`slrtfile_data` — Workspace variable containing real-time `Scope` file format data

vector

Data Types: `uint8`

Output Arguments

`matlab_data` — State and time data for plotting

structure

The state and time data is stored in a structure containing six fields. The key fields are numSignals, data, and signalNames.

`version` — Version code

`0` (default) | double

Internal

`sector` — Sector of data file

`0` (default) | double

Internal

`headersize` — Number of bytes of data file header

`512` (default) | double

Internal

`numSignals` — Number of columns containing signal and time data

double

If *N* signals are connected to the real-time Scope block, numSignals = `N + 1`.

`data` — Columns containing signal and time data

double array

The data array contains numSignals columns. The first *N* columns represent signal state data. The last column contains the time at which the state data is captured.

The data array contains as many rows as there are data points.

`signalNames` — Names of columns containing signal and time data

cell vector

The signalNames vector contains numSignals elements. The first *N* elements are signal names. The last element is the string `Time`.

See Also

Scope | `SimulinkRealTime.copyFileToHost` | `SimulinkRealTime.fileSystem`

Introduced in R2014a

Target Settings Properties

Store settings related to target computer

This object defines the settings for the target computer.

The settings define the properties of the communication link between the development and target computers and the properties of the target boot image created during the setup process.

To create a new target computer settings object, use the syntax `target_object = SimulinkRealTime.addTarget(target_name)`.

```
target_object = SimulinkRealTime.addTarget('TargetPC1')
```

```
Simulink Real-Time Target Settings

    Name                     : TargetPC1

    TargetRAMSizeMB          : Auto
    MaxModelSize             : 1MB
    SecondaryIDE             : off
    NonPentiumSupport        : off
    MulticoreSupport         : off
    LegacyMultiCoreConfig    : off
    USBSupport               : on
    ShowHardware             : off
    EthernetIndex            : 0

    HostTargetComm           : TcpIp
    TcpIpTargetAddress       :
    TcpIpTargetPort          : 22222
    TcpIpSubNetMask          : 255.255.255.0
    TcpIpGateway             : 255.255.255.255
    RS232HostPort            : COM1
    RS232Baudrate            : 115200
    TcpIpTargetDriver        : Auto
    TcpIpTargetBusType       : PCI
    TcpIpTargetISAMemPort    : 0x300
    TcpIpTargetISAIRQ        : 5

    TargetScope              : Enabled
    TargetBoot                  : BootFloppy
    BootFloppyLocation          :
```

To read existing target computer settings, use the syntax `value = target_object.property_name`.

```
target_object = SimulinkRealTime.getTargetSettings('TargetPC1');
value = target_object.HostTargetComm

value =

TcpIp
```

To change an existing setting by assignment, use the syntax `target_object.property_name = value`.

```
target_object = SimulinkRealTime.getTargetSettings('TargetPC1');
target_object.HostTargetComm = 'RS232';

value = target_object.HostTargetComm

value =

RS232
```

To mark a target computer as the default computer, use the syntax `setAsDefaultTarget(target_object)`.

```
target_object = SimulinkRealTime.getTargetSettings('TargetPC1');
setAsDefaultTarget(target_object)
```

To access the target computer settings in Simulink Real-Time Explorer:

1. In the **Targets** pane, expand a target computer node.
2. In the toolbar, click the Target Properties icon .
3. Expand the sections **Host-to-Target communication**, **Target settings**, or **Boot configuration**.

Host-to-Target Communication

`HostTargetComm` — Type of link between development and target computers

`'TcpIp'` (default) | `'RS232'`

Use 'TcpIp' if you have a dedicated Ethernet card installed in both the development and target computers. Use 'RS232' if you have a dedicated COM port in both the development and target computers.

Note: RS-232 communication type will be removed in a future release. Use TCP/IP instead.

In the Simulink Real-Time Explorer **Communication type** list, select one of RS-232 or TCP/IP.

If you select RS-232, you must also set the property RS232HostPort. If you select TCP/IP, then you must set the other properties that start with TcpIp.

Example: env_object.HostTargetComm = 'RS232'

RS232Baudrate — Serial link transmission rate

'115200' (default) | '57600' | '38400' | '19200' | '9600' | '4800 | '2400' | '1200'

In the Simulink Real-Time Explorer **Baud rate** list, select one of 1200, 2400, 4800, 9600, 19200, 38400, 57600, or 115200.

Before you can select a baud rate, you must set the HostTargetComm property to RS232.

Example: env_object.RS232Baudrate = '57600'

RS232HostPort — Serial link COM port on development computer

'COM1' (default) | 'COM2'

Selects serial port on development computer only. The software determines the COM port on the target computer.

In the Simulink Real-Time Explorer **Host port** list, select one of COM1 or COM2.

Before you can select an RS-232 port, you must set the HostTargetComm property to RS232.

Example: env_object.RS232HostPort = 'COM2'

TcpIpGateway — IP address for gateway to Ethernet link

'255.255.255.255' (default) | '*xxx.xxx.xxx.xxx*'

If you communicate with your target computer from within a LAN that uses gateways, and your development and target computers are connected through a gateway, you must enter a value for this property.

The default value, `255.255.255.255`, means that a gateway is not used to connect to the target computer. If your LAN does not use gateways, you do not need to change this property. Consult your system administrator for this value.

In the Simulink Real-Time Explorer **Gateway** box, type the IP address for your gateway.

Example: `env_object.TcpIpGateway = '192.168.1.1'`

`TcpIpSubNetMask` — Subnet mask for gateway to Ethernet link

`'`*`xxx.xxx.xxx.xxx`*`'`

In the Simulink Real-Time Explorer **Subnet mask** box, type the subnet mask of your LAN. Consult your system administrator for this value.

Example: `env_object.TcpIpSubNetMask = '255.255.255.0'`

`TcpIpTargetAddress` — IP address for target computer

`'`*`xxx.xxx.xxx.xxx`*`'`

In the Simulink Real-Time Explorer **IP address** box, type a valid IP address for your target computer. Consult your system administrator for this value.

Example: `env_object.TcpIpTargetAddress = '192.168.1.10'`

`TcpIpTargetBusType` — Bus type for Ethernet card on target computer

`'PCI'` (default) | `'ISA'` | `'USB'`

This property determines the bus type of your target computer. You do not need to define a bus type for your development computer.

If `TcpIpTargetBusType` is set to `PCI`, then the properties `TcpIpISAMemPort` and `TcpIpISAIRQ` are not used for TCP/IP communication.

If you are using an ISA bus card, set `TcpIpTargetBusType` to `ISA` and enter values for `TcpIpISAMemPort` and `TcpIpISAIRQ`.

In the Simulink Real-Time Explorer **Bus type** list, select one of `PCI`, `ISA`, or `USB`.

Example: `env_object.TcpIpTargetBusType = 'USB'`

TcpIpTargetDriver — Driver for Ethernet card on target computer

'Auto' (default) | '3C90x' | 'I8254x' | 'I82559' | 'NE2000' | 'NS83815' | 'R8139' | 'R8168' | 'Rhine' | 'RTLANCE' | 'SMC91C9X' | 'USBAX772' | 'USBAX172'

Use the default value ('Auto') if the target computer contains only one supported Ethernet card.

Use 'USBAX772' or 'USBAX172' if you are using bus type 'USB'.

In the Simulink Real-Time Explorer **Target driver** list, select one of THREECOM_3C90x, INTEL_I8254x, INTEL_I82559, NE2000, NS83815, R8139, R8168, Rhine, RTLANCE, SMC91C9X, USBAX772, USBAX172, or Auto.

Example: env_object.TcpIpTargetDriver = 'USBAX172'

TcpIpTargetISAIRQ — IRQ for Ethernet card on ISA bus target computer

'5' (default) | '*N*' | '15'

IRQ values run from '5' to '15', inclusive.

If you are using an ISA bus Ethernet card, you must enter a value for TcpIpISAIRQ. The value must correspond to the jumper or ROM settings on the ISA bus Ethernet card.

On your ISA bus card, assign an IRQ by moving the jumpers on the card. Set the IRQ to 5, 10, or 11. If one of these hardware settings leads to a conflict in your target computer, choose another IRQ and make the corresponding changes to your jumper settings.

From the Simulink Real-Time Explorer **IRQ** list, select an IRQ value.

Example: env_object.TcpIpTargetISAIRQ = '11'

TcpIpTargetISAMemPort — IRQ base address for Ethernet card on ISA bus target computer

0x*NNNN*

If you are using an ISA bus Ethernet card, you must enter a value for the property TcpIpISAMemPort. The value of this property must correspond to the jumper or ROM settings on your ISA bus Ethernet card.

On your ISA bus card, assign an II/O port base address by moving the jumpers on the card. Set the I/O port base address to a value near 0x300. If one of these hardware settings leads to a conflict in your target computer, choose another I/O port base address and make the corresponding changes to your jumper settings.

In the Simulink Real-Time Explorer **Address** box, type an I/O port base address.

Example: `env_object.TcpIpTargetISAMemPort = '0x400'`

`TcpIpTargetPort` — Ethernet port on target computer

`'22222'`. (default) | `'xxxxx'`

Use an Ethernet port address greater than `'20000'`. Values in this range are higher than the reserved area (`telnet`, `ftp`, . . .). This address is used only on the target computer.

You typically do not change this value from the default. You should only do so if you are using the default port (`'22222'`) for other purposes.

Example: `env_object.TcpIpTargetPort = '24000'`

Target settings

`EthernetIndex` — Index number of Ethernet card on target computer

`'0'` (default) | `'n'`

Unique number identifying an Ethernet card on the target computer. If the target computer has multiple Ethernet cards, you must select one of the cards for the Ethernet link. This option returns the index number of the card selected on the target computer upon starting.

The `(n-1)th` Ethernet card on the target computer has an index number `'n'`.

Example: `env_object.EthernetIndex = '2'`

`LegacyMultiCoreConfig` — Target computer contains legacy hardware

`'off'` (default) | `'on'`

Set this value to `'on'` only if your target computer contains hardware not compliant with the Advanced Configuration and Power Interface (ACPI) standard. Otherwise, leave this value set to `'off'`.

Example: `env_object.LegacyMultiCoreConfig = 'on'`

`MaxModelSize` — Maximum expected size of real-time application

`'1MB'` (default) | `'4MB'`

The maximum model size reserves the specified amount of memory on the target computer for the real-time application. Memory not used by the real-time application is used by the kernel and by the heap for data logging.

Selecting too high a value leaves less memory for data logging. Selecting too low a value does not reserve enough memory for the real-time application and creates an error. You can approximate the size of the real-time application by the size of the DLM file produced by the build process.

In the Simulink Real-Time Explorer **Model size** list, select one of `1 MB` or `4 MB`.

Setting **Model size** is enabled for **Boot mode** `Stand Alone` only. Value `'16MB'` is not supported.

Example: `env_object.MaxModelSize = '4MB'`

`MulticoreSupport` — Enable use of multicore processors

`'off'` (default) | `'on'`

Use multicore support only for a multicore target computer.

In the Simulink Real-Time Explorer , select the **Multicore CPU** check box to take advantage of these processors for background tasks. Otherwise, clear it.

Example: `env_object.MulticoreSupport = 'on'`

`Name` — Target computer name string

`'TargetPCN'` (default) | string

When you create a new target settings object, the software assigns it a name of the form `'TargetPCN+1'`, where `'TargetPCN'` is the previously assigned name. You can assign a new name from the Command Window.

To rename the target computer in Simulink Real-Time Explorer, right-click the target computer node in the **MATLAB Session** tree, click **Rename**, and type the new name in the **Target environment name** box.

Example: `env_object.Name = 'NewTarget'`

`NonPentiumSupport` — Target computer contains legacy processor

`'off'` (default) | `'on'`

Set only if your target computer has a 386 or 486 compatible processor. If your target computer has a Pentium or higher compatible processor, selecting this check box slows the performance of your target computer.

If your target computer has a 386 or 486 compatible processor, select the Simulink Real-Time Explorer **Target is a 386/486** check box. Otherwise, clear it.

Example: `env_object.NonPentiumSupport = 'on'`

`SecondaryIDE` — Enable secondary IDE disk controller

`'off'` (default) | `'on'`

Set only if you want to use disks connected to a secondary IDE controller.

To set this parameter in Simulink Real-Time Explorer, select the **Secondary IDE** check box. Otherwise, clear it.

Example: `env_object.SecondaryIDE = 'on'`

`ShowHardware` — Display Ethernet card information for target computer

`'off'` (default) | `'on'`

If you create a target boot kernel when `ShowHardware` is `'on'` and start the target computer with it, the kernel displays the index, bus, slot, function, and target driver for each Ethernet card on the target monitor.

The development computer cannot communicate with the target computer after the kernel starts with `ShowHardware` set. When you are done gathering the information provided by the kernel on when you start the target computer with `ShowHardware='on'`, you must set this property to `'off'`, recreate the boot image, and restart the target computer to resume normal functionality.

Example: `env_object.ShowHardware = 'on'`

`TargetRAMSizeMB` — Megabytes of RAM installed in target computer

`'Auto'` (default) | `'`*`xxx`*`'`

Specifies the total amount of RAM, in megabytes, installed on the target computer. Target computer RAM is used for the kernel, real-time application, data logging, and other functions that use the heap.

If this property is set to `'Auto'`, the real-time application reads the target computer BIOS and determines the amount of memory up to a maximum of 4 GB.

To allow the real-time application to determine the amount of memory in Simulink Real-Time Explorer, click **RAM size Auto**. If the real-time application cannot read the BIOS, click **Manual** and type into the **Size(MB)** box the amount of RAM, in megabytes, installed on the target computer.

The Simulink Real-Time kernel can use only 4 GB of memory.

Example: `env_object.ShowHardware = '2000'`

`TargetScope` — Display scope information graphically

`'Enabled'` (default) | `'Disabled'`

When this property is set to `'Enabled'`, the target computer shows a graphical windowed display. When set to `'Disabled'`, the target computer shows a text-based view.

When the graphical display is present, you can use target scopes to view signal data graphically on the target display. You cannot do this when the text-based view is present.

Using Simulink Real-Time Explorer, to display scope information graphically, set the **Graphics mode** check box.

To display scope information as text, clear the **Graphics mode** check box.

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

Example: `env_object.TargetScope = 'Disabled'`

`USBSupport` — Enable USB port on target computer

`'on'` (default) | `'off'`

Set this property to use a USB port on the target computer, for example to connect a USB mouse.

In Simulink Real-Time Explorer, to enable a USB port, select the **USB Support** check box. Otherwise, clear it.

Example: `env_object.USBSupport = 'off'`

Boot configuration

`BootFloppyLocation` — Drive name for creation of target boot disk

string

Set this property if you need to create a removable boot disk and the system default drive does not work.

Example: `env_object.BootFloppyLocation='D:\'`

`DOSLoaderLocation` — Location of DOS Loader files to start target computers from devices other than floppy disk or CD

string

Set this property in DOS Loader mode if the default location does not work.

Example: `env_object.DOSLoaderLocation='D:\Dosloader'`

`TargetBoot` — Mode of restarting target computer

`'BootFloppy'` (default) | `'CDBoot'` | `'DOSLoader'` | `'NetworkBoot'` | `'StandAlone'`

After making the required target settings, to create a bootable image, type `SimulinkRealTime.createTargetImage`.

In Simulink Real-Time Explorer, to create a bootable image for the specified boot mode, click **Create boot disk**.

Example: `env_object.TargetBoot='NetworkBoot'`

`TargetMACAddress` — Target computer MAC address for network restart

`'`*`xx:xx:xx:xx:xx:xx`*`'`

Physical target computer MAC address from which to accept start requests when starting within a dedicated network.

To update the MAC address in Simulink Real-Time Explorer, first click the **Reset** button in the **Target Properties** pane. You can then click the **Specify new MAC address** button to enter a MAC address manually in the **MAC address** box. If you do not enter a MAC address manually, the software will obtain the MAC address automatically the next time you restart the target computer.

Example: `env_object.TargetMACAddress='90:e2:ba:17:5d:15'`

See Also

`SimulinkRealTime.addTarget` | `SimulinkRealTime.getTargetSettings` | `SimulinkRealTime.targetSettings.setAsDefaultTarget`

Related Examples

- “Ethernet Link Setup”
- “Serial Link Setup”

- “Target Computer Settings”
- “Target Boot Methods”

Introduced in R2014a

SimulinkRealTime.targetSettings.setAsDefaultTarget

Set specific target computer environment object as default

Syntax

```
setAsDefaultTarget(target_object)
```

Description

`setAsDefaultTarget(target_object)` sets the specified target computer as the default target computer from the `SimulinkRealTime.target` class.

Examples

Set target computer `'TargetPC1'` as the default target computer:

```
target_object = SimulinkRealTime.getTargetSettings('TargetPC1');
setAsDefaultTarget(target_object)
```

Introduced in R2014a

SimulinkRealTime.fileSystem

Manage folders and files on target computer

Description

This class implements folder and file access methods used on the target computer.

Constructor

Constructor	Description
`SimulinkRealTime.fileSy (constructor)`	Create file system object

Methods

Method	Description
`SimulinkRealTime.fileSy`	Change folder on target computer
`SimulinkRealTime.fileSy`	List contents of current folder on target computer
`SimulinkRealTime.fileSy`	Information about target computer drive
`SimulinkRealTime.fileSy`	Close open target computer file or files
`SimulinkRealTime.fileSy`	Target computer file information
`SimulinkRealTime.fileSy`	Information about open files in target computer file system
`SimulinkRealTime.fileSy`	Open target computer file for reading
`SimulinkRealTime.fileSy`	Read open target computer file
`SimulinkRealTime.fileSy`	Write binary data to open target computer file
`SimulinkRealTime.fileSy`	Size of file on target computer
`SimulinkRealTime.fileSy`	Make folder on target computer
`SimulinkRealTime.fileSy`	Current folder path of target computer
`SimulinkRealTime.fileSy`	Remove file from target computer
`SimulinkRealTime.fileSy`	Remove folder from target computer

Introduced in R2014a

SimulinkRealTime.fileSystem (constructor)

Create Simulink Real-Time file system object

Syntax

```
filesys_object = SimulinkRealTime.fileSystem
filesys_object = SimulinkRealTime.fileSystem(target_object)
```

Arguments

`filesys_object`	Variable name to reference the file system object.
`target_object`	Variable name to reference the target object.

Description

Constructor of a SimulinkRealTime.fileSystem object. The file system object represents the target computer file system. You work with the target computer file system from the development computer using file system methods.

`filesys_object = SimulinkRealTime.fileSystem` returns the file system object corresponding to the default target. Use this form if you have one target computer or if you designate a target computer as the default one in your system.

`filesys_object = SimulinkRealTime.fileSystem(target_object)` returns the file system object corresponding to the target computer accessible by `target_object`.

Examples

Create a file system object for the default target computer:

```
fsys = SimulinkRealTime.fileSystem
```

If you have a `SimulinkRealTime.target` object, you can construct a `SimulinkRealTime.fileSystem` object by passing the `SimulinkRealTime.target` object variable to the `SimulinkRealTime.fileSystem` constructor as an argument:

```
tg = SimulinkRealTime.target('TargetPC1');
fsys = SimulinkRealTime.fileSystem(tg)
```

See Also

`SimulinkRealTime.fileSystem`

Introduced in R2014a

SimulinkRealTime.fileSystem.cd

Change folder on target computer

Syntax

```
cd(file_obj, target_computer_dir)
```

Arguments

`file_obj`	Name of the `SimulinkRealTime.fileSystem` object.
`target_computer_dir`	Name of the target computer folder to change.

Description

From the development computer, `cd(file_obj, target_computer_dir)` changes the currently active folder on the target computer.

Examples

For the file system object `fsys`, change the folder from the current one to one named `'logs'`:

```
tg = slrt;
fsys = SimulinkRealTime.fileSystem(tg);
cd(fsys,'logs')
```

See Also

`cd` | `SimulinkRealTime.fileSystem.mkdir` | `SimulinkRealTime.fileSystem.pwd`

Introduced in R2014a

SimulinkRealTime.fileSystem.dir

List contents of current folder on target computer

Syntax

```
dir(file_obj)
dir(file_obj,folder_name)
return_value = dir(file_obj,___)
```

Arguments

`file_obj`	Handle of the file system object.
`folder_name`	Name of a folder on the target computer.
`return_value`	Struct array returned from the `SimulinkRealTime.fileSystem` object, consisting of the following fields: • `date` — The last date at which the object was saved. • `time` — The last time at which the object was saved. • `isdir` — If `1`, the object is a folder. If `0`, it is not a folder. • `bytes` — Size in bytes of that object. • `name` — Name of an object in the folder, shown as a cell array. The name, stored in the first element of the cell array, can have up to eight characters. The three-character file extension is stored in the second element of the cell array.

Description

From the development computer, `dir(file_obj)` lists the contents of the currently active folder on the target computer.

From the development computer, `dir(file_obj,folder_name)` lists the contents of the folder `folder_name` on the target computer.

`return_value = dir(file_obj,___)` returns the results in an M-by-1 structure.

Examples

List the contents of the currently active folder for the file system object `fsys`:

```
tg = slrt;
fsys = SimulinkRealTime.fileSystem(tg);
dir(fsys)
```

```
4/12/1998     20:00             222390        IO  SYS
 11/2/2003    13:54                  6    MSDOS  SYS
 11/5/1998    20:01              93880  COMMAND  COM
 11/2/2003    13:54  <DIR>           0     TEMP
 11/2/2003    14:00                 33 AUTOEXEC  BAT
  11/2/2003   14:00                512 BOOTSECT  DOS
  18/2/2003   16:33               4512 SC1SIGNA  DAT
 18/2/2003    16:17  <DIR>           0    FOUND  000
 29/3/2003    19:19               8512     DATA  DAT
 28/3/2003    16:41               8512 DATADATA  DAT
 28/3/2003    16:29               4512 SC4INTEG  DAT
  1/4/2003     9:28          201326592 PAGEFILE  SYS
 11/2/2003    14:13  <DIR>           0    WINNT
    4/5/2001  13:05             214432 NTLDR      '
  4/5/2001    13:05              34468 NTDETECT  COM
 11/2/2003    14:15  <DIR>           0  DRIVERS
  22/1/2001   11:42                217   BOOT    INI'
 28/3/2003    16:41               8512        A  DAT
 29/3/2003    19:19               2512 SC3SIGNA  DAT
 11/2/2003    14:25  <DIR>           0  INETPUB
 11/2/2003    14:28                  0   CONFIG  SYS
 29/3/2003    19:10               2512 SC3INTEG  DAT
  1/4/2003    18:05               2512  SC1GAIN  DAT
   11/2/2003  17:26  <DIR>           0 UTILIT~1
```

You must use the `dir(f)` syntax to list the contents of the folder.

Return the contents of the DOS directory as a struct array:

```
return_value = dir(file_obj,'DOS')
```

```
return_value =
1x12 struct array with fields:

    date
    time
    isdir
    bytes
    name
```

See Also

dir | SimulinkRealTime.fileSystem.mkdir | SimulinkRealTime.fileSystem.cd | SimulinkRealTime.fileSystem.pwd

Introduced in R2014a

SimulinkRealTime.fileSystem.diskinfo

Target computer drive configuration information

Syntax

```
return_value = diskinfo(filesys_obj,target_computer_drive)
```

Arguments

filesys_obj	Name of the SimulinkRealTime.fileSystem file system object.
target_computer_drive	Name of the target computer drive being accessed.

Description

return_value = diskinfo(filesys_obj,target_computer_drive) is called from the development computer and returns configuration information for the specified drive on the target computer.

Examples

For file system object fsys, return configuration information for the target computer C:\ drive:

```
tg = slrt;
fsys = SimulinkRealTime.fileSystem(tg);
diskinfo(fsys,'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
```

Introduced in R2014a

SimulinkRealTime.fileSystem.fclose

Close target computer file

Syntax

```
fclose(filesys_obj,file_id)
```

Arguments

`filesys_obj`	Name of the `SimulinkRealTime.fileSystem` file system object
`file_id`	File identifier of the file to close

Description

From the development computer, `fclose(filesys_obj,file_id)` closes one or more open files in the target computer file system (except standard input, output, and error). The `file_id` argument is the file identifier associated with an open file. You cannot have more than eight files open at the same time in the file system.

Examples

Close the open file identified by the file identifier `h` in the file system object `fsys`:

```
tg = slrt;
fsys = SimulinkRealTime.fileSystem(tg);
h = fopen(fsys, 'data.dat', 'w');
fwrite(fsys, h, 'test')
fclose(fsys, h)
h = fopen(fsys, 'data.dat', 'r');
value = fread(fsys, h);
char(value)
```

See Also

`fclose` | `SimulinkRealTime.fileSystem.fopen` | `SimulinkRealTime.fileSystem.fread` | `SimulinkRealTime.fileSystem.filetable` | `SimulinkRealTime.fileSystem.fwrite`

Introduced in R2014a

SimulinkRealTime.fileSystem.fileinfo

Target computer file configuration information

Syntax

```
return_value = fileinfo(filesys_obj,file_id)
```

Arguments

filesys_obj	Name of the SimulinkRealTime.fileSystem file system object.
file_id	Identifier of the file for which to get file configuration information.

Description

From the development computer, `return_value = fileinfo(filesys_obj,file_id)` gets file configuration information for the file on the target computer associated with `file_id`.

Examples

Return file configuration information for the target computer file associated with the file identifier `h` in the file system object `fsys`:

```
tg = slrt;
fsys = SimulinkRealTime.fileSystem(tg);
h = fopen(fsys, 'data.dat', 'r');
fileinfo(fsys,h)
```

```
ans =

          FilePos: 0
    AllocatedSize: 32768
```

```
          ClusterChains: 1
     VolumeSerialNumber: 1082284597
                FulName: 'C:\data.dat'
```

Introduced in R2014a

SimulinkRealTime.fileSystem.filetable

Information about open files in target computer file system

Syntax

```
return_value = filetable(filesys_obj,file_id)
```

Arguments

`filesys_obj`	Name of the `SimulinkRealTime.fileSystem` file system object.

Description

Method of `SimulinkRealTime.fileSystem` objects. From the development computer, `return_value = filetable(filesys_obj,file_id)` returns a table of the open files in the target computer file system. You cannot have more than eight files open at the same time in the file system.

Note: Use the `filetable` function only to recover the lost file handle value when MATLAB exits with files still open on the target computer. The function has no other use.

Examples

Return a table of the open files in the target computer file system for the file system object `fsys`:

```
tg = slrt;
fsys = SimulinkRealTime.fileSystem(tg);
filetable(fsys)
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 hexadecimal handle to a handle that other `SimulinkRealTime.fileSystem` methods can use, use the MATLAB `hex2dec` function:

```
h1 = hex2dec('001E0001'))
h1 =
1966081
```

To close that file, use `SimulinkRealTime.fileSystem.fclose`.

```
fclose(fsys,h1);
```

See Also

`SimulinkRealTime.fileSystem.fopen` | `SimulinkRealTime.fileSystem.fclose` | `hex2dec`

Introduced in R2014a

SimulinkRealTime.fileSystem.fopen

Open target computer file for reading

Syntax

```
file_id = fopen(file_obj, file_name)
file_id = fopen(file_obj, file_name, permission)
```

Arguments

`file_obj`	Name of the `SimulinkRealTime.fileSystem` object.
`file_name`	Name of the target computer to open, in single quotes
`permission`	Permission values, one of `'r'`, `'w'`, `'a'`, `'r+'`, `'w+'`, or `'a+'`.
`file_id`	Identifier for newly-opened file.

The permission values have the following meaning.:

- `'r'` — Open the file for reading (default). If the file does not already exist, the method does not do anything.
- `'w'` — Open the file for writing. If the file does not already exist, the method creates the file.
- `'a'` — Open the file for appending to it. Initially, the file pointer is at the end of the file. If the file does not already exist, the method creates the file.
- `'r+'` — Open the file for reading and writing. Initially, the file pointer is at the beginning of the file. If the file does not already exist, the method does not do anything.
- `'w+'` — Open the file for reading and writing. If the file exists, the method empties the file and places the file pointer at the beginning of the file. If the file does not already exist, the method creates the file.
- `'a+'` — Open the file for reading and appending to the file. Initially, the file pointer is at the end of the file. If the file does not already exist, the method creates the file.

Description

From the development computer, `file_id = fopen(file_obj, file_name)` opens the specified file name on the target computer for reading binary data.

`file_id = fopen(file_obj, file_name, permission)` opens the specified file name on the target computer for reading binary data.

You cannot have more than eight files open at the same time 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 computer file system object `fsys` and read the file using the resulting file handle:

```
tg = slrt;
fsys = SimulinkRealTime.fileSystem(tg);
h = fopen(fsys,'data.dat')

ans =

     2883584

d = fread(fsys,h);
```

See Also

`fopen` | `SimulinkRealTime.fileSystem.fclose` | `SimulinkRealTime.fileSystem.fread` | `SimulinkRealTime.fileSystem.fwrite`

Introduced in R2014a

SimulinkRealTime.fileSystem.fread

Read open target computer file

Syntax

```
data = fread(file_obj,file_id)
data = fread(file_obj,file_id,offset,numbytes)
```

Arguments

`file_obj`	Name of the `SimulinkRealTime.fileSystem` object.
`file_id`	File identifier of the file to read.
`numbytes`	Maximum number of bytes `fread` can read.
`offset`	The position, measured from the beginning of the file, from which `fread` can start to read.
`data`	Matrix containing the binary data read.

Description

`data = fread(file_obj,file_id)` reads binary data from the file on the target computer and writes it into matrix `data`. The `file_id` argument is the file identifier associated with an open file.

`data = fread(file_obj,file_id,offset,numbytes)` reads `numbytes` bytes from `file_id` starting from position `offset` and writes the block into matrix `data`.

To get a count of the total number of bytes read into `data`, use the following:

```
count = length(data);
```

`length(data)` can be less than `numbytes` if that number of bytes is not available. `length(data)` is zero if `fread` is positioned at the end of the file.

Examples

Open the file data.dat in the target computer file system object fsys and read the file using the resulting file handle:

```
tg = slrt;
fsys = SimulinkRealTime.fileSystem(tg);
h = fopen(fsys,'data.dat')
d = fread(fsys,h);
```

This function reads the file data.dat and stores the contents of the file to d. This content is in the Simulink Real-Time file format.

See Also

fread | SimulinkRealTime.fileSystem.fclose | SimulinkRealTime.fileSystem.fopen | SimulinkRealTime.fileSystem.fwrite

Introduced in R2014a

SimulinkRealTime.fileSystem.fwrite

Write binary data to open target computer file

Syntax

```
fwrite(file_obj,file_id,data)
```

Arguments

`file_obj`	Name of the `SimulinkRealTime.fileSystem` object.
`file_id`	File identifier of the file to write.
`data`	Elements of matrix `data` to write to the specified file.

Description

From the development computer, `fwrite(file_obj,file_id,data)` writes the elements of matrix `data` 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. `fwrite` requires that the file be open with write permission.

Examples

Open the file `data.dat` in the target computer file system object `fsys` and write the file using the resulting file handle:

```
tg = slrt;
fsys = SimulinkRealTime.fileSystem(tg);
h = fopen(fsys,'data.dat','w');
fwrite(fsys,h,magic(5));
```

This command writes the elements of matrix `magic(5)` to the file handle `h`. This content is written in column order.

See Also

`fwrite` | `SimulinkRealTime.fileSystem.fclose` | `SimulinkRealTime.fileSystem.fopen` | `SimulinkRealTime.fileSystem.fread`

Introduced in R2014a

SimulinkRealTime.fileSystem.getfilesize

Size of file on target computer

Syntax

```
file_size = getfilesize(file_obj,file_id)
```

Arguments

`file_obj`	Name of the `SimulinkRealTime.fileSystem` object
`file_id`	File identifier of the file being sized
`file_size`	Number of bytes in the file

Description

From the development computer, `file_size = getfilesize(file_obj,file_id)` gets the size (in bytes) of the file identified by the `file_id` file identifier on the target computer file system. Use the Simulink Real-Time file object method `fopen` to open the file system object.

Examples

Get the size of the file identifier `h` for the file system object `fsys`:

```
tg = slrt;
fsys = SimulinkRealTime.fileSystem(tg);
getfilesize(fsys,h)
```

See Also

`SimulinkRealTime.fileSystem.fopen`

Introduced in R2014a

SimulinkRealTime.fileSystem.mkdir

Create folder on target computer

Syntax

```
mkdir(file_obj,dir_name)
```

Arguments

`file_obj`	Name of the `SimulinkRealTime.fileSystem` object.
`dir_name`	Name of the folder to be created.

Description

From the development computer, `mkdir(file_obj,dir_name)` makes a new folder in the current folder on the target computer file system.

Examples

Create a new folder, `logs`, in the target computer file system object `fsys`:

```
tg = slrt;
fsys = SimulinkRealTime.fileSystem(tg);
mkdir(fsys,'logs')
```

See Also

`mkdir` | `SimulinkRealTime.fileSystem.dir` | `SimulinkRealTime.fileSystem.pwd`

Introduced in R2014a

SimulinkRealTime.fileSystem.pwd

Path to currently active folder on target computer

Syntax

```
active_folder = pwd(file_obj)
```

Arguments

`file_obj`	Name of the `SimulinkRealTime.fileSystem` object.
`active_folder`	Path to the currently active folder on the target computer.

Description

Called from the development computer, `active_folder = pwd(file_obj)` returns the path to the currently active folder on the target computer. Unless `cd(file_obj, target_computer_dir)` has been called, the currently active folder is the top folder of the boot drive, usually `C:\`.

Examples

Return the currently active folder for the file system object `fsys`:

```
tg = slrt;
fsys = SimulinkRealTime.fileSystem(tg);
pwd(fsys)
```

See Also

`pwd` | `SimulinkRealTime.fileSystem.cd` | `SimulinkRealTime.fileSystem.dir` | `SimulinkRealTime.fileSystem.mkdir`

Introduced in R2014a

SimulinkRealTime.fileSystem.removefile

Remove file from target computer

Syntax

```
removefile(file_obj,file_name)
```

Arguments

`file_name`	Name of the file to remove from the target computer file system.
`file_obj`	Name of the `SimulinkRealTime.fileSystem` object.

Description

Called from the development computer, `removefile(file_obj,file_name)` removes a file from the target computer file system.

Note: You cannot recover this file once you remove it

Examples

Remove the `data2.dat` file from the target computer file system `fsys`:

```
tg = slrt;
fsys = SimulinkRealTime.fileSystem(tg);
removefile(fsys,'data2.dat')
```

Introduced in R2014a

SimulinkRealTime.fileSystem.rmdir

Remove folder from target computer

Syntax

```
rmdir(file_obj,dir_name)
```

Arguments

`dir_name`	Name of the folder to remove from the target computer file system.
`file_obj`	Name of the `SimulinkRealTime.fileSystem` object.

Description

Called from the development computer, `rmdir(file_obj,dir_name)` removes a folder from the target computer file system.

Note: You cannot recover this folder once you remove it.

Examples

Remove the `data2dir.dat` folder from the target computer file system `fsys`:

```
tg = slrt;
fsys = SimulinkRealTime.fileSystem(tg);
rmdir(fsys,'data2dir.dat')
```

Introduced in R2014a

SimulinkRealTime.fileSystem.selectdrive

Select target computer drive

Syntax

```
selectdrive(file_obj,'drive')
```

Arguments

`drive`	Name of the drive to set.
`file_obj`	Name of the `SimulinkRealTime.fileSystem` object.

Description

Called from the development computer, `selectdrive(file_obj,'drive')` sets the currently active drive of the target computer to the specified string. Enter the drive string with an extra backslash (\). For example, `D:\\` for the D:\ drive.

Examples

Set the current target computer drive to D:\:

```
tg = slrt;
fsys = SimulinkRealTime.fileSystem(tg);
selectdrive(fsys,'D:\\')
```

Introduced in R2014a

Using Real-Time Application Objects

Represent real-time application and target computer status

Object represents currently loaded real-time application and target computer status.

Object provides access to methods and properties that start and stop the real-time application, read and set parameters, monitor signals, and retrieve status information about the target computer. You can also reboot the target computer and load and unload the real-time application.

Function 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.

Some of the object properties and functions can be invoked from the target computer command line when the real-time application has been loaded.

Examples

Build and run real-time application

Build and download `xpcosc`, execute real-time application in external mode

Open, build, and download real-time application

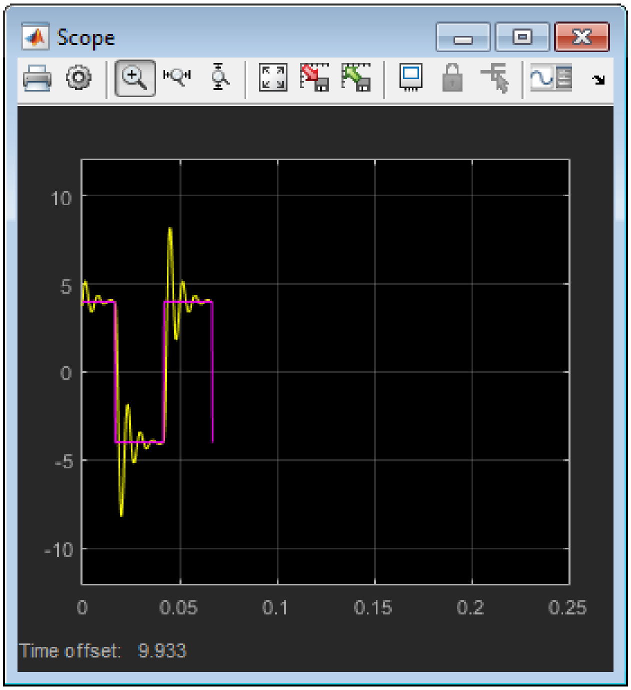
```
ex_model = 'xpcosc';
open_system(ex_model);
ex_scope = [ex_model '/Scope'];
open_system(ex_scope)
rtwbuild(ex_model);
tg = SimulinkRealTime.target
```

```
Target: TargetPC1
   Connected            = Yes
   Application          = xpcosc
   Mode                 = Real-Time Single-Tasking
   Status               = stopped
   CPUOverload          = none
   ExecTime             = 0.0000
   SessionTime          = 794.4953
   StopTime             = 0.200000
   SampleTime           = 0.000250
   AvgTET               = NaN
   MinTET               = Inf
   MaxTET               = 0.000000
   ViewMode             = 0

   TimeLog              = Vector(0)
   StateLog             = Matrix (0 x 2)
   OutputLog            = Matrix (0 x 2)
   TETLog               = Vector(0)
   MaxLogSamples        = 16666
   NumLogWraps          = 0
   LogMode              = Normal

   Scopes               = No Scopes defined
   NumSignals           = 7
   ShowSignals          = off

   NumParameters        = 7
   ShowParameters       = off
```

Prepare and run simulation in external mode for 10 seconds

```
tg.StopTime = 10;
set_param(ex_model,'SimulationMode','External');
set_param(ex_model,'SimulationCommand','Connect');
set_param(ex_model,'SimulationCommand','Start');
pause(10);
set_param(ex_model,'SimulationCommand','Stop');
set_param(ex_model,'SimulationCommand','Disconnect');
```

The output looks like this:



Unload real-time application

```
unload(tg)
```

```
Target: TargetPC1
   Connected            = Yes
   Application          = loader
```

Properties

Real-Time Application Properties

Object Functions

`SimulinkRealTime.target.ping` `SimulinkRealTime.target.reboot` `SimulinkRealTime.target.load` `SimulinkRealTime.target.unload` `SimulinkRealTime.target.close` `SimulinkRealTime.target.start` `SimulinkRealTime.target.stop`

`SimulinkRealTime.target.addscope` `SimulinkRealTime.target.getscope`
`SimulinkRealTime.target.remscope` `SimulinkRealTime.target.getlog`
`SimulinkRealTime.target.getsignal` `SimulinkRealTime.target.getsignalid`
`SimulinkRealTime.target.getsignalidsfromlabel`
`SimulinkRealTime.target.getsignallabel`
`SimulinkRealTime.target.getsignalname`
`SimulinkRealTime.target.getparam` `SimulinkRealTime.target.setparam`
`SimulinkRealTime.target.getparamid`
`SimulinkRealTime.target.getparamname`
`SimulinkRealTime.target.loadparamset`
`SimulinkRealTime.target.saveparamset`

Create Object

`SimulinkRealTime.target`

See Also

"Target Computer Commands"

More About

- "Blocks Whose Outputs Depend on Inherited Sample Time"

Introduced in R2014a

Real-Time Application Properties

Properties of real-time application and target computer

Provides access to the properties of the real-time application and the target computer.

To get the value of a readable target object property from a target object:

```
value = target_object.property_name
```

For example, to get the `CommunicationTimeOut` of the target object:

```
target_object = slrt;
value = target_object.CommunicationTimeOut
```

To set the value of a writable target object property from a target object:

```
target_object.property_name = new_value
```

For example, to set the `CommunicationTimeOut` of the target object:

```
target_object = slrt;
target_object.CommunicationTimeOut = 10
```

At the target computer command line, you can set the target object properties `stoptime`, `sampletime`, and writable model parameters.

```
stoptime = floating_point_number
sampletime = floating_point_number
setpar parameter_index = parameter_value
```

Target Computer

`Application` — Name of real-time application

`'loader'` | string

Name of real-time application running on target computer, specified as a string. This is the name of the Simulink model from which the application was built. When the target computer starts, this value is `'loader'`.

`CommunicationTimeOut` — Communication timeout between development and target computers

5 (default) | seconds

Communication timeout between the development and target computers, specified in seconds.

`Connected` — Communication status between development and target computers

`'No'` (default) | `'Yes'`

Communication status between the development and target computers, specified as string.

`CPUoverload` — CPU status for overload

`'none'` (default) | `'detected'`

CPU status for overload, specified as string. If the real-time application requires more CPU time than the sample time of the model, the kernel changes this value from `'none'` to `'detected'` and stops the current run. To keep this status `'none'` you must user a faster processor or specify a larger sample time.

`Mode` — Execution mode of the real time application

`'Real-Time Singletasking'` (default) | `'Real-Time Multitasking'`

Execution mode of the real time application on the target computer, specified as a string. The execution mode is governed by parameter settings during Simulink Coder code generation.

`SessionTime` — Time since kernel started running on target computer

seconds

Time since the kernel started running on the target computer, specified in seconds. This time is also the elapsed time since you started the target computer.

Real-Time Execution

`AvgTET` — Average task execution time

seconds

Average task execution time, specified in seconds. This value is an average of the measured CPU times required to run the model equations and post outputs during each sample interval.

Task execution time is nearly constant, with minor deviations due to cache, memory access, interrupt latency, and multirate model execution.

The TET includes:

- Complete I/O latency.
- Data logging for output, state, and TET, as well as the data captured in scopes.
- Time spent executing tasks related to asynchronous interrupts while the real time task is running.
- Parameter updating latency. This latency is incurred if the **Double buffer parameter changes** parameter is set in the **Simulink Real-Time Options** node of the model Configuration Parameters dialog box.

The TET is not the only consideration in determining the minimum achievable sample time. Other considerations are:

- Time required to measure TET.
- Interrupt latency required to schedule and run one step of the model.

`ExecTime` — Execution time of real-time application

seconds

Execution time of real-time application since your real-time application started running, specified in seconds. When the real-time application stops, the kernel displays the total execution time.

`MaxTET` — Maximum task execution time

seconds

Maximum task execution time, specified in seconds. Corresponds to the slowest time (longest measured time) required to update model equations and post outputs.

`MinTET` — Minimum task execution time

seconds

Minimum task execution time, specified in seconds. Corresponds to the fastest time (smallest measured time) required to update model equations and post outputs.

`SampleTime` — Time between samples (step size)

seconds

Time between samples (step size), in seconds, for updating the model equations and posting the outputs.

`Status` — Execution status of real-time application

`'stopped'` (default) | `'running'`

Execution status of real-time application, specified as string.

`StopTime` — Time when real-time application stops running

seconds | `'Inf'`

Time when the real-time application stops running, specified in seconds or as string. The initial value is set in the **Solver** pane of the Configuration Parameters dialog box.

When the `ExecTime` reaches `StopTime`, the application stops running. If you specify the special value `'Inf'`, the real-time application runs until you manually stop it or restart the target computer.

`TETLog` — Storage in the MATLAB workspace for task execution time vector

vector of double

Storage in the MATLAB workspace for task execution time vector, specified as a vector of double.

Signal Visualization

`LogMode` — Controls which data points are logged

`'Normal'` (default) | double

Controls which data points are logged, as specified by the keyword `'Normal'` or a double.

- `'Normal'` — Indicates time-equidistant logging. Logs a data point at every time interval.
- Double — Indicates value-equidistant logging. Logs a data point only when an output signal from the `OutputLog` changes by the specified difference in signal value (increment).

`MaxLogSamples` — Maximum number of samples for each logged signal

unsigned integer

Maximum number of samples for each logged signal, specified as an unsigned integer.

`NumLogWraps` — Number of times the circular data logging buffer wraps

unsigned integer

Number of times the circular data logging buffer wraps, specified as an unsigned integer. The buffer wraps each time the number of samples exceeds `MaxLogSamples`.

`NumSignals` — Number of observable signals

unsigned integer

Number of observable signals in Simulink model, specified as an unsigned integer. Nonobservable signals, such as structures, are not included in this value.

`OutputLog` — Storage in MATLAB workspace for output or Y-vector

matrix

Storage in MATLAB workspace for output or Y-vector, specified as a matrix.

`Scopes` — List of index numbers, one per scope

vector of unsigned integer

List of index numbers, one per scope, specified as a vector of unsigned integers.

`ShowSignals` — Flag set to display the list of signals

`'off'` (default) | `'on'`

Flag set to view the list of signals from your Simulink model, specified as string. MATLAB displays the signal list when you display the properties for a target object.

`Signals` — List of observable signals

vector of structures

List of observable signals, specified as a vector containing the following values for each signal:

- Property name — `S0`, `S1`. . .
- Property value — Value of the signal.
- Block name— Hierarchical name of the Simulink block that the signal comes from.

This list is visible only when `ShowSignals` is set to `'on'`.

`StateLog` — Storage in MATLAB workspace for state or X-vector

matrix

Storage in MATLAB workspace for state or X-vector, specified as a matrix.

`TimeLog` — Storage in the MATLAB workspace for time or T-vector

vector of double

Storage in the MATLAB workspace for time or T-vector, specified as a vector of double.

`ViewMode` — Display specified scope or all scopes on target computer

`0` (default) | unsigned integer | `'all'`

Display specified scope or all scopes on target computer. This property is active only if the environment property `TargetScope` is set to `enabled`.

Parameter Tuning

`NumParameters` — Number of tunable parameters

unsigned integer

Number of tunable parameters in Simulink model, specified as an unsigned integer. Nontunable (nonobservable) parameters, such as structures, are not included in this value.

`Parameters` — List of tunable parameters

vector of structures

List of tunable parameters, specified as a vector containing the following values for each parameter:

- 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 the parameter in a Simulink block.
- Block name — If the parameter is a block parameter, this is the hierarchical name of the Simulink block containing the parameter. If the parameter is a model parameter, this is the empty string.

This list is visible only when `ShowParameters` is set to `'on'`.

`ShowParameters` — Flag set to display the list of parameters

`'off'` (default) | `'on'`

Flag set to view the list of parameters from your Simulink model, specified as string. MATLAB displays the parameter list when you display the properties for a target object.

See Also

“Target Computer Commands” | Using Real-Time Application Objects

Introduced in R2014a

SimulinkRealTime.target

Create object representing real-time application on target computer

Syntax

```
target_object = SimulinkRealTime.target
target_object = SimulinkRealTime.target(target_name)
```

Description

`target_object = SimulinkRealTime.target` constructs a target object representing the default target computer.

When MATLAB evaluates the return value on the development computer, it attempts to connect to the target computer. If the attempt succeeds, MATLAB prints `Connected = Yes`, followed by the status of the application running on the target computer. If the attempt fails, MATLAB waits until the connection times out, and then prints `Connected = No`. To avoid the timeout delay, verify that the target computer is operational and connected to the development computer, or suppress output with a terminating semicolon.

`target_object = SimulinkRealTime.target(target_name)` constructs a target object representing the target computer designated by `target_name`.

Examples

Default Target Computer

Create a target object that communicates with the default target computer. Report the status of the default target computer. In this case, the target computer is connected to the development computer and is executing the loader.

```
target_object = SimulinkRealTime.target
```

```
Target: TargetPC1
   Connected            = Yes
   Application          = loader
```

Specific Target Computer

Create a target object that communicates with target computer `TargetPC1`. Report the status of the target computer. In this case, the target computer is not connected to the development computer.

```
target_object = SimulinkRealTime.target('TargetPC1')
```

```
Target: TargetPC1
   Connected            = No
```

Input Arguments

`target_name` — Name assigned to target computer
string

Example: `'TargetPC1'`

Data Types: `char`

Output Arguments

`target_object` — Object representing target computer
object variable

Object of type `SimulinkRealTime.target` that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required the Ethernet link settings.

Example: `tg`

Data Types: `struct`

See Also

Using Real-Time Application Objects | Real-Time Application Properties | `slrt` | Target Settings Properties

Introduced in R2014a

SimulinkRealTime.target.addscope

Create a scope of specified type

Syntax

```
scope_object = addscope(target_object)
scope_object = addscope(target_object, scope_type, scope_number)
scope_object_vector = addscope(target_object, scope_type,
scope_number_vector)
```

Description

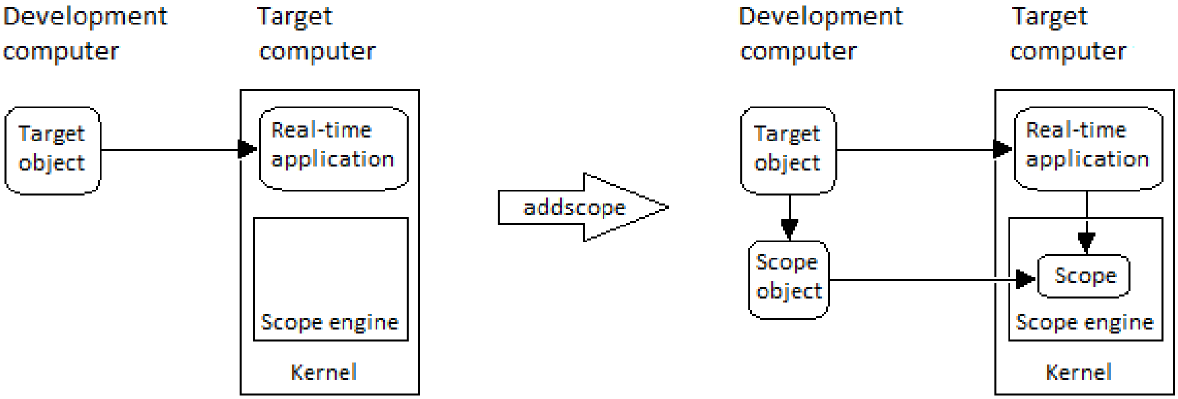
`scope_object = addscope(target_object)` creates on the target computer a host scope, assigns as its scope number the next available integer in the target object property `Scopes`, and returns the object representing this scope.

`scope_object = addscope(target_object, scope_type, scope_number)` creates on the target computer a scope of the given type with the given scope number and returns the object representing this scope.

`scope_object_vector = addscope(target_object, scope_type, scope_number_vector)` creates on the target computer a set of scopes of the given type with the given scope numbers and returns a vector of objects representing these scopes.

`addscope` updates the target object property `Scopes`. If the result is not assigned to a MATLAB variable, the scope object properties are listed in the Command Window.

The Simulink Real-Time product supports 9 target scopes, 8 file scopes, and as many host scopes as the target computer resources can support. If you try to add a scope with the same index as an existing scope, the result is an error.



At the target computer command line, you can add a single target scope:

```
addscope
addscope scope_number
```

Examples

Create default scope with default number

Create a default (host) scope with the default (next available) number and assign it to sc1

```
tg = slrt;
sc1 = addscope(tg)
```

```
sc1 =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 1
   Status               = Interrupted
   Type                 = Host
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = -1
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 1
   TriggerSample        = 0
   StartTime            = -1.000000
   Data                 = Matrix (250 x 0)
   Time                 = Matrix (250 x 1)
   Signals              = no Signals defined
```

Create file scope number 2

Create a file scope with number 2 and assign it to sc2.

```
tg = slrt;
sc2 = addscope(tg,'file',2)
```

```
sc2 =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 2
   Status               = Interrupted
   Type                 = File
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = -1
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 2
   TriggerSample        = 0
   FileName             = unset
   WriteMode            = Lazy
   WriteSize            = 512
   AutoRestart          = off
   DynamicFileName      = off
   MaxWriteFileSize     = 536870912
   Signals              = no Signals defined
```

Create vector of target scopes numbers 3 and 4

Create two target scopes 3 and 4 using a vector of scope numbers and assign the scope objects to variable scvector.

```
tg = slrt;
scope_object_vector = addscope(tg, 'target', [3, 4])

scope_object_vector =

Simulink Real-Time Scope
   Application          = xpco   ScopeId              = 3
   Status               = Interrupted
   Type                 = Target
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = -1
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 3
   TriggerSample        = 0
   DisplayMode          = Redraw (Graphical)
   YLimit               = Auto
   Grid                 = on
   Signals              = no Signals defined

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 4
   Status               = Interrupted
   Type                 = Target
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = -1
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 4
   TriggerSample        = 0
   DisplayMode          = Redraw (Graphical)
   YLimit               = Auto
   Grid                 = on
   Signals              = no Signals defined
```

Input Arguments

`target_object` — Object representing target computer

object variable

Object of type `SimulinkRealTime.target` that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required the Ethernet link settings.

Example: `tg`

Data Types: `struct`

`scope_type` — Type of scope to create

`'host'` (default) | `'target'` | `'file'`

Type of scope to create, as a string. This argument is optional. The default value is `'host'`.

`scope_number` — New scope number

`unsigned integer`

New scope number. This argument is optional. The default value is the next available integer in the target object property `Scopes`.

If you enter the scope number for an existing scope object, the result is an error.

Example: `1`

`scope_number_vector` — Vector of new scope numbers

`unsigned integer vector`

Vector of new scope numbers. If you enter the scope number for an existing scope object, the result is an error.

Example: `[2, 3]`

Output Arguments

`scope_object` — Object representing newly-created scope

object

Object representing the newly-created scope

`scope_object_vector` — Vector of objects representing newly-created scope

object

Vector containing objects representing the newly-created scope

See Also

"Target Computer Commands" | Using Real-Time Application Objects | Real-Time Application Properties | `SimulinkRealTime.target.getscope` | `SimulinkRealTime.target.remscope`

Introduced in R2014a

SimulinkRealTime.target.close

Close connection between development and target computers

Syntax

```
status_string = close(target_object)
```

Description

`status_string = close(target_object)` closes the connection between the development computer and a target computer. The target object and other associated objects are still valid, and will automatically connect to the target computer the next time they are accessed.

Examples

Close communication with target computer `'TargetPC1'`

Access target computer `'TargetPC1'` and close the connection

Get a target object for target computer `'TargetPC1'`

```
tg = SimulinkRealTime.target('TargetPC1')
```

```
Target: TargetPC1
   Connected            = Yes
   Application          = loader
```

Close the connection

```
close(tg)
```

```
ans =
Communication is closed
```

Input Arguments

`target_object` — Object representing target computer
object variable

Object of type `SimulinkRealTime.target` that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required the Ethernet link settings.

Example: `tg`

Data Types: `struct`

Output Arguments

`status_string` — Report results of attempt to close communication
`'Communication is closed'`

Returns literal string on every call, if `close` did not fail.

See Also

Using Real-Time Application Objects | Real-Time Application Properties | `SimulinkRealTime.target` | `SimulinkRealTime.target.reboot`

Introduced in R2014a

SimulinkRealTime.target.getLastErrorMessage

Get last error message printed on target screen

Syntax

```
getLastErrorMessage(target_object)
message_string = getLastErrorMessage(target_object)
```

Description

`getLastErrorMessage(target_object)` displays the last error message printed on the target screen. If an error has not occurred, the function displays the empty string.

`message_string = getLastErrorMessage(target_object)` returns the last error message printed on the target screen. If an error has not occurred, the function returns the empty string.

Examples

Display and Return Last Sample Time Error Message

Set `xpcosc` sample time to a small value. Display and return error message

Open and build `xpcosc`. Get the target object. Read the sample time.

```
ex_model = 'xpcosc';
open_system(ex_model);
rtwbuild(ex_model);
target_object = slrt;
target_object.SampleTime
```

```
ans =

   2.5000e-04
```

Set the sample time to `0.0000001`.

```
target_object.SampleTime = 0.0000001
```

```
TargetPC1: Invalid value for Ts (must be between 8e-6 and 10 for
interrupt mode, or between 5e-7 and 10 for polling mode)
```

Display the last error message.

```
getLastErrorMessage(target_object)
```

```
ans =

Error: sample time must be in the range of [8e-006, 10]
```

Return the last error message.

```
message_string = getLastErrorMessage(target_object)
```

```
message_string =

Error: sample time must be in the range of [8e-006, 10]
```

Unload and close model.

```
unload(target_object);
close_system(ex_model);
```

Input Arguments

`target_object` — Object representing target computer

object variable

Object of type `SimulinkRealTime.target` that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required the Ethernet link settings.

Example: `tg`

Data Types: `struct`

Output Arguments

`message_string` — Last error message printed on target screen

string

Introduced in R2015a

SimulinkRealTime.target.getlog

Portion of output logs from target object

Syntax

```
log = getlog(target_object, log_name)
log = getlog(target_object, log_name, first_point)
log = getlog(target_object, log_name, first_point, number_samples)
log = getlog(target_object, log_name, first_point, number_samples,
decimation)
```

Description

`log = getlog(target_object, log_name)` returns all the samples from a log of type `log_name`, starting from the first point without decimation.

`log = getlog(target_object, log_name, first_point)` returns the sample at `first_point` from a log of type `log_name`.

`log = getlog(target_object, log_name, first_point, number_samples)` returns `number_samples` samples from a log of type `log_name`, starting from `first_point` without decimation.

`log = getlog(target_object, log_name, first_point, number_samples, decimation)` returns `number_samples` samples from a log of type `log_name`, starting from `first_point`, with decimation `decimation`.

Examples

Retrieve all values

Read the `TimeLog` and `OutputLog` samples from model `xpcosc` using the default settings. Plot the results.

Read `TimeLog` and `OutputLog` samples

```
tg = slrt;
timelog = getlog(tg, 'TimeLog');
outputlog = getlog(tg, 'OutputLog');
```

Plot the data

```
plot(timelog, outputlog);
```

Retrieve 10 values starting from 5

Read 10 samples starting from 5 of TimeLog and OutputLog

Read 5 TimeLog samples

```
tg = slrt;
timelog = getlog(tg, 'TimeLog', 5, 10)
```

```
timelog =

    0.0010
    0.0013
    0.0015
    0.0018
    0.0020
    0.0023
    0.0025
    0.0027
    0.0030
    0.0033
```

Read 10 OutputLog samples

```
outputlog = getlog(tg, 'OutputLog', 5, 10)
```

```
outputlog =

   -1.6200   -4.0000
   -2.3450   -4.0000
   -3.0990   -4.0000
   -3.8345   -4.0000
   -4.5098   -4.0000
   -5.0907   -4.0000
   -5.5518   -4.0000
   -5.8772   -4.0000
   -6.0606   -4.0000
   -6.1046   -4.0000
```

Plot the data

```
plot(timelog, outputlog);
```

Retrieve 10 values starting from 5 with decimation 2

Read 10 samples at decimation 2 starting from 5 of `TimeLog` and `OutputLog`

Read 5 `TimeLog` samples

```
tg = slrt;
timelog = getlog(tg, 'TimeLog', 5, 10, 2)
```

```
timelog =

    0.0010
    0.0015
    0.0020
    0.0025
    0.0030
    0.0035
    0.0040
    0.0045
    0.0050
    0.0055
```

Read 10 `OutputLog` samples

```
outputlog = getlog(tg, 'OutputLog', 5, 10, 2)
```

```
 -1.6200   -4.0000
   -3.0990   -4.0000
   -4.5098   -4.0000
   -5.5518   -4.0000
   -6.0606   -4.0000
   -6.0199   -4.0000
   -5.5384   -4.0000
   -4.8028   -4.0000
   -4.0224   -4.0000
   -3.3784   -4.0000
```

Plot the data

```
plot(timelog, outputlog);
```

Retrieve 1 value

Read 1 sample starting from sample 8 of TimeLog and OutputLog

Read 5 TimeLog samples

```
tg = slrt;
timelog = getlog(tg, 'TimeLog', 8)
```

```
timelog =

    0.0018
```

Read 10 OutputLog samples

```
outputlog = getlog(tg, 'OutputLog', 8)
```

```
outputlog =

   -3.8345   -4.0000
```

- “Set Configuration Parameters”

Input Arguments

target_object — Object representing target computer

object variable

Object of type SimulinkRealTime.target that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required the Ethernet link settings.

Example: tg

Data Types: struct

log_name — Selects information type to retrieve

'TimeLog' | 'StateLog' | 'OutputLog' | 'TETLog'

- TimeLog — Time stamps for each logged value
- StateLog — Discrete and continuous state of blocks

- OutputLog — Value of root-level outport blocks
- TETLog — Task execution times (TET)

Example: 'Timelog'

Data Types: char

first_point — Sample from which to start retrieving data

1 (default) | positive integer

If specified without number_samples, this parameter returns only the value at first_point.

Example: 10

number_samples — Number of samples to retrieve

all points in log (default) | positive integer

Number of samples to retrieve starting with first_point, after decimation.

Example: 10

decimation — Select every decimationth value

1 (default) | positive integer

1 returns all sample points. n returns every nth sample point. Must be used with first_point and number_samples.

Example: 2

Output Arguments

log — User-defined MATLAB variable

matrix

Variable receives the log entries as a matrix

Introduced in R2014a

SimulinkRealTime.target.getparam

Read value of observable parameter in real-time application

Syntax

```
value = getparam(target_object, parameter_index)
```

Description

`value = getparam(target_object, parameter_index)` returns the value of the parameter associated with `parameter_index`.

Examples

Get block parameter value given names of parameter and block

Get the value of block parameter `'Gain'` of block `'Gain'`

Get block parameter index from parameter hierarchical name

```
tg = slrt;
pid = getparamid(tg, 'Gain', 'Gain');
```

Get value of block parameter

```
getparam(tg, pid)
```

```
ans =

     1000000
```

Get model parameter value given name of parameter

Get the value of model parameter `'G2'`

Get model parameter index from parameter name

```
tg = slrt;
pid = getparamid(tg, '', 'G2');
```

Get value of model parameter

```
getparam(tg, pid)
```

```
ans =

     1000000
```

Input Arguments

`target_object` — Object representing target computer
object variable

Object of type `SimulinkRealTime.target` that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required the Ethernet link settings.

Example: `tg`

Data Types: `struct`

`parameter_index` — Index number of the parameter
nonnegative integer

The parameter index can mark either a block parameter or a model parameter. To be accessible via parameter index, the parameter must be observable.

Example: `0`

Example: `1`

Output Arguments

`value` — Value of parameter
number | string

Structure parameters are not observable.

More About

- “Parameter Tuning Basics”

- “Nonobservable Signals and Parameters”

See Also

Using Real-Time Application Objects | Real-Time Application Properties | `SimulinkRealTime.target.getparamid` | `SimulinkRealTime.target.setparam`

Introduced in R2014a

SimulinkRealTime.target.getparamid

Parameter index from parameter hierarchical name

Syntax

```
parameter_index = getparamid(target_object, block_name,
parameter_name)
parameter_index = getparamid(target_object, '', parameter_name)
```

Description

`parameter_index = getparamid(target_object, block_name, parameter_name)` returns the index of a block parameter in the parameter list based on the path to the parameter name. You must enter the names in full. The names are case sensitive.

For `block_name`, enter the mangled name that Simulink Coder uses for code generation.

`parameter_index = getparamid(target_object, '', parameter_name)` returns the index of a model parameter in the parameter list based on the parameter name. The name is case sensitive.

For the second (block name) argument, enter the empty string (`''`).

Examples

Get block parameter value given names of parameter and block

Get the value of block parameter `'Gain'` of block `'Gain'`

Get block parameter index from parameter hierarchical name

```
tg = slrt;
pid = getparamid(tg, 'Gain', 'Gain');
```

Get value of block parameter

```
getparam(tg, pid)
```

```
ans =

     1000000
```

Get model parameter value given name of parameter

Get the value of model parameter `'G2'`

Get model parameter index from parameter name

```
tg = slrt;
pid = getparamid(tg, '', 'G2');
```

Get value of model parameter

```
getparam(tg, pid)
```

```
ans =

     1000000
```

Input Arguments

`target_object` — Object representing target computer

object variable

Object of type `SimulinkRealTime.target` that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required the Ethernet link settings.

Example: `tg`

Data Types: `struct`

`block_name` — Hierarchical name of the originating block

string | `''`

The empty string (`''`) marks a model parameter, which is not associated with a particular block.

Example: `'Gain'`

Example: `''`

`parameter_name` — Name of block or model parameter

string

The parameter index can mark either a block parameter or a model parameter.

Example: `'Gain'`

Example: `'G2'`

Output Arguments

`parameter_index` — Index number of the parameter

nonnegative integer

The parameter index can mark either a block parameter or a model parameter. To be accessible via parameter index, the parameter must be observable.

Example: `0`

Example: `1`

More About

- “Parameter Tuning Basics”
- “Nonobservable Signals and Parameters”

See Also

Using Real-Time Application Objects | Real-Time Application Properties | `SimulinkRealTime.target.getparam` | `SimulinkRealTime.target.setparam`

Introduced in R2014a

SimulinkRealTime.target.getparamname

Block path and parameter name from parameter index

Syntax

```
[block_path, parameter_name] = getparamname(target_object,
parameter_index)
```

Arguments

`target_object`	Name of a target object.
`parameter_index`	Index number of the parameter.
`[block_path, parameter_name]`	Output vector containing the block path and parameter name for the specified parameter.

Description

`[block_path, parameter_name] = getparamname(target_object, parameter_index)` returns a vector containing two strings, block path and parameter name, for the parameter specified by `parameter_index`.

Examples

Get the block path and parameter name of parameter index 5:

```
tg = slrt;
[block_path, parameter_name] = getparamname(tg,5)
block_path =
Signal Generator
parameter_name =
Amplitude
```

Introduced in R2014a

SimulinkRealTime.target.getPCIInfo

Return information about PCI boards installed in target computer

Syntax

```
getPCIInfo(target_object, 'installed')
getPCIInfo(target_object,'ethernet')
getPCIInfo(target_object, 'all')
getPCIInfo(target_object, 'verbose')
pci_devices = getPCIInfo(target_object, ___)

getPCIInfo(target_object, 'supported')
getPCIInfo(target_object, 'supported', 'ethernet')
pci_devices_supported = getPCIInfo(target_object, 'supported', ___)
```

Description

`getPCIInfo(target_object, 'installed')` queries the target computer, represented by `target_object`, for installed PCI devices (boards) that are supported by driver blocks in the Simulink Real-Time block library. The software displays in the Command Window information about the PCI devices that `getPCIInfo` found, including:

- PCI bus number
- Slot number
- Assigned IRQ number
- Vendor (manufacturer) name
- Device (board) name
- Device type
- Vendor PCI ID
- Device PCI ID
- Device release version

Before you can use this call, you must verify that the target computer has started properly under the Simulink Real-Time kernel and that the Ethernet link is working. The real-time application can be loaded or the loader can be active and waiting for input. You can check these preconditions by verifying that the function `SimulinkRealTime.pingTarget` returns `success`.

Before building the model, you can use `getPCIInfo` to find resources to enter into a driver block dialog box. Such resources include PCI bus number, slot number, and assigned IRQ number.

`getPCIInfo(target_object,'ethernet')` queries the target computer, represented by `target_object`, for installed Ethernet controllers supported by Simulink Real-Time.

`getPCIInfo(target_object, 'all')` displays information about all of the PCI devices found on the target computer represented by `target_object`. This information includes graphics controllers, network cards, SCSI cards, and devices that are part of the motherboard chip set (for example, PCI-to-PCI bridges).

`getPCIInfo(target_object, 'verbose')` shows the information displayed by `getPCIInfo(target_object, 'all')` for the target computer represented by `target_object`, plus information about the PCI addresses that the BIOS assigns to this board.

`pci_devices = getPCIInfo(target_object, ___)` queries the target computer represented by `target_object` according to the additional arguments you supplied. The call returns a structure containing information about the PCI devices that the software found on the target computer.

`getPCIInfo(target_object, 'supported')` displays a list of the PCI devices supported by the Simulink Real-Time block library. This call does not access the target computer, so the Ethernet link does not have to be active.

`getPCIInfo(target_object, 'supported', 'ethernet')` displays a list of the Ethernet controllers that are supported by Simulink Real-Time. This call does not access the target computer, so the Ethernet link does not have to be active.

`pci_devices_supported = getPCIInfo(target_object, 'supported', ___)` returns a structure containing a list of devices supported by Simulink Real-Time according to the additional arguments you supplied. This call does not access the target computer, so the Ethernet link does not have to be active.

Examples

Display information for PCI devices on default computer that the Simulink Real-Time block library supports

Start the default target computer with the Simulink Real-Time kernel. Verify the connection between the development and the target computer. At the command prompt, type the command on the development computer.

```
slrtpingtarget

target_object = slrt;
getPCIInfo(target_object, 'installed')

List of installed PCI devices:

Measurement Computing    PCI-DIO24
     Bus 1, Slot 11, IRQ 10
     DI DO
     VendorID 0x1307, DeviceID 0x0028,
          SubVendorID 0x1307, SubDeviceID 0x0028
     A/D Chan: 0, D/A Chan: 0, DIO Chan: 24
     Released in: R14SP2 or Earlier

.
.
.
```

Display information for Ethernet controllers on default computer that Simulink Real-Time supports

Start the default target computer with the Simulink Real-Time kernel. Verify the connection between the development and target computers. At the MATLAB command prompt, type the command on the development computer.

```
slrtpingtarget

target_object = slrt;
getPCIInfo(target_object, 'ethernet')

List of installed PCI devices:

Intel                    82541GI_LF
```

```
     Bus 16, Slot 4, IRQ 10
     Ethernet controller
     VendorID 0x8086, DeviceID 0x107c, SubVendorID 0x8086,
         SubDeviceID 0x1376
     Released in: R2006b
     Notes: Intel Gigabit Ethernet series
```

Display information for all PCI devices on default computer

Start the default target computer with the Simulink Real-Time kernel. Verify the connection between the development and target computers. At the command prompt, type the command on the development computer.

```
slrtpingtarget

target_object = slrt;
getPCIInfo(target_object, 'all')
```

```
List of installed PCI devices:

Intel                    Unknown
     Bus 0, Slot 0, IRQ 0
     Host Bridge
     VendorID 0x8086, DeviceID 0x1130,
          SubVendorID 0x8086, SubDeviceID 0x4532
.
.
.
Measurement Computing    PCI-DIO24
     Bus 1, Slot 11, IRQ 10
     DI DO
     VendorID 0x1307, DeviceID 0x0028,
          SubVendorID 0x1307, SubDeviceID 0x0028
     A/D Chan: 0, D/A Chan: 0, DIO Chan: 24
     Released in: R14SP2 or Earlier
.
.
.
```

Display verbose information for all PCI devices on default computer

Start the default target computer with the Simulink Real-Time kernel. Verify the connection between the development and target computers. At the command prompt, type the command on the development computer.

```
slrtpingtarget

target_object = slrt;
getPCIInfo(target_object, 'verbose')

List of installed PCI devices:

Intel                    Unknown
     Bus 0, Slot 0, IRQ 0
     Host Bridge
     VendorID 0x8086, DeviceID 0x1130,
          SubVendorID 0x8086, SubDeviceID 0x4532
     BaseClass 6, SubClass 0
     BAR BaseAddress AddressSpace   MemoryType PreFetchable
      0)    E8000000       Memory   32-bit decoder       no
.
.
.
Measurement Computing    PCI-DIO24
     Bus 1, Slot 11, IRQ 10
     DI DO
     VendorID 0x1307, DeviceID 0x0028,
          SubVendorID 0x1307, SubDeviceID 0x0028
     A/D Chan: 0, D/A Chan: 0, DIO Chan: 24
     Released in: R14SP2 or Earlier
     BaseClass FF, SubClass FF
     BAR BaseAddress AddressSpace
      1)        DC00          I/O
      2)        DFF4          I/O
.
.
.
```

Return information for PCI devices on default computer that the Simulink Real-Time block library supports

Start the default target computer with the Simulink Real-Time kernel. Verify the connection between the development and target computers. At the command prompt, type the command on the development computer. Display the first structure in the vector.

```
slrtpingtarget

target_object = slrt;
```

```
pci_devices = getPCIInfo(target_object);
pci_devices(1)
```

```
ans =

              Bus: 1
             Slot: 11
         VendorID: '1307'
         DeviceID: '28'
      SubVendorID: '1307'
      SubDeviceID: '28'
        BaseClass: 'FF'
         SubClass: 'FF'
        Interrupt: 10
    BaseAddresses: [1x6 struct]
       VendorName: 'Measurement Computing'
          Release: 'R14SP2 or Earlier'
            Notes: ''
       DeviceName: 'PCI-DIO24'
       DeviceType: 'DI DO'
           ADChan: '0'
           DAChan: '0'
          DIOChan: '24'
```

Return information for all PCI devices on default computer

Start the default target computer with the Simulink Real-Time kernel. Verify the connection between the development and target computers. At the command prompt, type the command on the development computer. Display the first structure in the vector.

```
slrtpingtarget
```

```
target_object = slrt;
pci_devices = getPCIInfo(target_object, 'all');
pci_devices(1)
```

```
ans =

              Bus: 0
             Slot: 0
         VendorID: '8086'
         DeviceID: '1130'
      SubVendorID: '8086'
      SubDeviceID: '4532'
```

```
        BaseClass: '6'
         SubClass: '0'
        Interrupt: 0
    BaseAddresses: [1x6 struct]
       VendorName: 'Intel'
          Release: ''
            Notes: ''
       DeviceName: 'Unknown'
       DeviceType: 'Host Bridge'
           ADChan: ''
           DAChan: ''
          DIOChan: ''
```

Return verbose information for all PCI devices via `target_object`

Start the default target computer with the Simulink Real-Time kernel. Use `SimulinkealTime.target` to get the `target_object`. Verify the connection between the development and target computers. At the command prompt, type the command on the development computer. Display the first structure in the vector.

```
SimulinkRealTime.pingTarget('TargetPC1')

pci_devices = getPCIInfo(target_object,'verbose');
pci_devices(1)
```

```
ans =

              Bus: 0
             Slot: 0
         VendorID: '8086'
         DeviceID: '1130'
      SubVendorID: '8086'
      SubDeviceID: '4532'
        BaseClass: '6'
         SubClass: '0'
        Interrupt: 0
    BaseAddresses: [1x6 struct]
       VendorName: 'Intel'
          Release: ''
            Notes: ''
       DeviceName: 'Unknown'
       DeviceType: 'Host Bridge'
           ADChan: ''
           DAChan: ''
          DIOChan: ''
```

Display information for all PCI devices that the Simulink Real-Time block library supports

At the command prompt, type the commands on the development computer. The target computer does not have to be active.

```
target_object = SimulinkRealTime.target
```

```
getPCIInfo(target_object, 'supported')
```

```
List of supported PCI devices:

Vendor                    Device             Type...

ADLINK                    PCI-6208A          AO DI DO...
B&B Electronics (Quatech) DSCP-200/300 (PXI) Serial Ports...
.
.
.
Speedgoat                 IO321 (PMC-FPGA)   AI (IO321-5)...
Speedgoat                 IO331 (PMC-FPGA)   DI DO (LVDS/LVCMOS)...
```

Display information for all Ethernet controllers that Simulink Real-Time supports

At the MATLAB prompt, type the command on the development computer.

```
target_object = SimulinkRealTime.target
```

```
getPCIInfo(target_object, 'supported', 'ethernet')
```

```
List of supported Ethernet controllers:

Vendor              Device       VendorID DeviceID Release

3Com                3c900B Combo 10B7     9005     R2006a+
3Com                3c905B Combo 10B7     9058     R2006a+
.
.
.
Winbond Electronics 89C940       1050     5A5A     R2006a+
Winbond Electronics 89C940       8C4A     1980     R2006a+
```

Return information for all PCI devices that the Simulink Real-Time block library supports

At the command prompt, type the commands on the development computer. The target computer does not have to be active.

```
target_object = SimulinkRealTime.target

pci_devices_supported = getPCIInfo(target_object, 'supported');
pci_devices_supported(1)

ans = 

       VendorID: '144A'
       DeviceID: '6208'
    SubVendorID: '-1'
    SubDeviceID: '-1'
     DeviceName: 'PCI-6208A'
     VendorName: 'ADLINK'
     DeviceType: 'AO DI DO'
         DAChan: '8'
         ADChan: '0'
        DIOChan: '4'
        Release: 'R14SP2 or Earlier'
          Notes: 'PCI-6208A features 8 current outputs w...'
```

Return information for all Ethernet controllers that Simulink Real-Time supports

At the MATLAB prompt, type the command on the development computer.

```
target_object = SimulinkRealTime.target

pci_devices_supported = 
      getPCIInfo(target_object, 'supported', 'ethernet');
pci_devices_supported(1)

ans = 

       VendorID: '10B7'
       DeviceID: '9005'
    SubVendorID: '-1'
    SubDeviceID: '-1'
     DeviceName: '3c900B Combo'
     VendorName: '3Com'
     DeviceType: 'Ethernet controller'
         DAChan: ''
         ADChan: ''
        DIOChan: ''
        Release: 'R2006a+'
          Notes: '3Com Etherlink 90x series'
```

- “Where to Find PCI Board Information”

- “Command-Line Ethernet Card Selection by Index”

Input Arguments

`target_object` — Object representing target computer

object variable

Object of type `SimulinkRealTime.target` that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required the Ethernet link settings.

Example: `tg`

Data Types: `struct`

Output Arguments

`pci_devices` — Information about the PCI devices in the target computer

vector

The vector that `getPCIInfo` returns when you call it without an argument contains information only for those PCI devices that the Simulink Real-Time library blocks support.

The vectors returned by `getPCIInfo` with the arguments `'all'` and `'verbose'` contain information about all PCI devices in the target computer. The vector are identical.

The fields in this structure are:

`Bus` — PCI bus where device resides

scalar

`Bus` and `Slot` uniquely identify the location of a device or bus adapter in the target computer.

`Slot` — PCI slot where device resides

scalar

`Slot` and `Bus` uniquely identify the location of a device or bus adapter in the target computer.

`VendorID` — Identifier for manufacturer of the device

string

Hexadecimal numeric string containing the identifier that the PCI standards organization assigns to the manufacturer of this device or bus adapter.

`DeviceID` — Identifier for device among those manufactured by the vendor

string

Hexadecimal numeric string containing the identifier that the manufacturer assigns to this device or bus adapter.

`SubVendorID` — Identifier for manufacturer of subsystem

string

Hexadecimal numeric string containing the identifier that the PCI standards organization assigns to the manufacturer of the entire subsystem (board).

`SubDeviceID` — Identifier for subsystem among those manufactured by the subvendor

string

Hexadecimal numeric string containing the identifier that the manufacturer assigns to this subsystem (board).

`BaseClass` — Standard PCI class of the device

string

Hexadecimal numeric string containing the standard PCI base classification of this device or bus adapter. `BaseClass` and `SubClass` identify the type and function of the device.

`SubClass` — Standard PCI subclass of the device

string

Hexadecimal numeric string containing the standard PCI subclass classification of this device or bus adapter. `SubClass` and `BaseClass` identify the type and function of the device.

`Interrupt` — IRQ used by the device

scalar

Provides the board-level interrupt that the device or bus adapter uses to trigger I/O with the target computer CPU.

`BaseAddresses` — Information for each Base Address Register (BAR) used by the device

vector

For each BAR used that this device or bus adapter uses, the vector contains a structure with the following fields:

`AddressSpaceIndicator` — Indicates whether the address is a memory or I/O address

0 | 1

- `0` — Memory address
- `1` — I/O address

`BaseAddress` — Memory address used by the device

string

Hexadecimal string containing the base memory address that the device uses.

`MemoryType` — Indicates the size of the address decode, 32-bit or 64-bit

0 | 1

Not used if `AddressSpaceIndicator` is `1` (I/O address).

- `0` — 32-bit address decode
- `1` — 64-bit address decode

`Prefetchable` — Indicates whether the memory is prefetchable

0 | 1

Not used if `AddressSpaceIndicator` is `1` (I/O address).

- `0` — Address is not prefetchable
- `1` — Address is prefetchable

`VendorName` — Name of vendor of device

string

Identifies the vendor of the specific device or bus adapter. Set to `'Unknown'` for unknown devices or bus adapters.

`Release` — MATLAB release version in which driver became available

string

If the Simulink Real-Time block library supports the device, it contains the MATLAB and Simulink release version in which the driver was released. Otherwise, it contains an empty vector.

`Notes` — Additional information about the device

string

Contains additional description of the device or bus adapter.

`DeviceName` — Name of device

string

Identifies the specific device or bus adapter. Set to `'Unknown'` for unknown devices or bus adapters.

`DeviceType` — Identifies the functions of the device

string

Contains abbreviations such as `'DI'` (digital input) that indicate the function or functions of the device or bus adapter.

`ADChan` — Number of analog inputs

string

Decimal numeric string containing the number of analog inputs to the device.

`DAChan` — Number of analog outputs

string

Decimal numeric string containing the number of analog outputs from the device.

`DIOChan` — Number of digital inputs and outputs

string

Decimal numeric string containing the number of digital inputs and outputs to and from the device.

`pci_devices_supported` — Information about the PCI devices supported by the product

vector

Vector of information about the devices and bus adapters that the blocks in the Simulink Real-Time block library represent.

The fields are as follows:

`VendorID` — Identifier for manufacturer of the device

string

Hexadecimal numeric string containing the identifier that the PCI standards organization assigns to the manufacturer of this device or bus adapter.

`DeviceID` — Identifier for device among those manufactured by the vendor

string

Hexadecimal numeric string containing the identifier that the manufacturer assigns to this device or bus adapter.

`SubVendorID` — Identifier for manufacturer of subsystem

string

Hexadecimal numeric string containing the identifier that the PCI standards organization assigns to the manufacturer of the entire subsystem (board).

`SubDeviceID` — Identifier for subsystem among those manufactured by the subvendor

string

Hexadecimal numeric string containing the identifier that the manufacturer assigns to this subsystem (board).

`DeviceName` — Name of device

string

Identifies the specific device or bus adapter. Set to `'Unknown'` for unknown devices or bus adapters.

`VendorName` — Name of vendor of device

string

Identifies the vendor of the specific device or bus adapter. Set to `'Unknown'` for unknown devices or bus adapters.

`DeviceType` — Identifies the functions of the device

string

Contains abbreviations such as `'DI'` (digital input) that indicate the function or functions of the device or bus adapter.

`DAChan` — Number of analog outputs

string

Decimal numeric string containing the number of analog outputs from the device.

`ADChan` — Number of analog inputs

string

Decimal numeric string containing the number of analog inputs to the device.

`DIOChan` — Number of digital inputs and outputs

string

Decimal numeric string containing the number of digital inputs and outputs to and from the device.

`Release` — MATLAB release version in which driver became available

string

If the Simulink Real-Time block library supports the device, it contains the MATLAB and Simulink release version in which the driver was released. Otherwise, it contains an empty vector.

`Notes` — Additional information about the device

string

Contains additional description of the device or bus adapter.

More About

- “PCI Bus I/O Devices”

See Also

Using Real-Time Application Objects | Real-Time Application Properties

Introduced in R2014a

SimulinkRealTime.target.getscope

Return scope identified by scope number

Syntax

```
scope_object_vector = getscope(target_object)
scope_object = getscope(target_object, scope_number)
scope_object_vector = getscope(target_object, scope_number_vector)
```

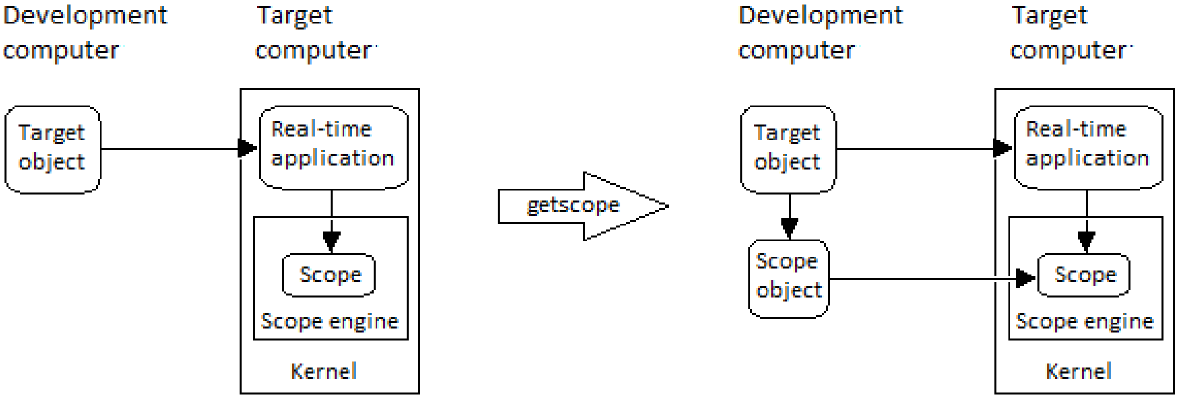
Description

`scope_object_vector = getscope(target_object)` returns a vector containing objects representing all of the existing scopes on the target computer.

`scope_object = getscope(target_object, scope_number)` returns the object representing an existing scope that has the given scope number.

`scope_object_vector = getscope(target_object, scope_number_vector)` returns a vector containing objects representing existing scopes that have the given scope numbers.

If you try to get a nonexistent scope, the result is an error.



Examples

All scopes on the target computer

To view the properties of allscopes on the target, get a vector of scope objects.

Get all scopes on the target computer.

```
tg = slrt;
scope_object_vector = getscope(tg)
```

```
scope_object_vector =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 1
   Status               = Interrupted
   Type                 = Target
   NumSamples           = 500
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 5  : Signal Generator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 1
   TriggerSample        = 0
   DisplayMode          = Redraw (Graphical)
   YLimit               = Auto
   Grid                 = on
   Signals              = 5  : Signal Generator
                          6  : Sum

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 2
   Status               = Interrupted
   Type                 = Target
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 0  : Gain
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 2
   TriggerSample        = 0
   DisplayMode          = Redraw (Graphical)
   YLimit               = Auto
   Grid                 = on
   Signals              = 0  : Gain
                          1  : Gain1
                          2  : Gain2

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 3
   Status               = Interrupted
   Type                 = Host
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = -1
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 3
   TriggerSample        = 0
   StartTime            = -1.000000
   Data                 = Matrix (250 x 0)
   Time                 = Matrix (250 x 1)
   Signals              = no Signals defined
```

Change the number of samples

To change the number of samples, get a scope object, and then change the scope object property NumSamples.

Get a scope object for scope 1.

```
tg = slrt;
scope_object = getscope(tg,1)
```

```
scope_object =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 1
   Status               = Interrupted
   Type                 = Target
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 5  : Signal Generator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 1
   TriggerSample        = 0
   DisplayMode          = Redraw (Graphical)
   YLimit               = Auto
   Grid                 = on
   Signals              = 5  : Signal Generator
                          6  : Sum
```

Update property `NumSamples`.

```
scope_object.NumSamples = 500
```

```
scope_object =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 1
   Status               = Interrupted
   Type                 = Target
   NumSamples           = 500
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 5  : Signal Generator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 1
   TriggerSample        = 0
   DisplayMode          = Redraw (Graphical)
   YLimit               = Auto
   Grid                 = on
   Signals              = 5  : Signal Generator
                          6  : Sum
```

Vector of scope objects

To view the properties of scopes 1 and 2 on the target computer, get a vector of scope objects.

```
tg = slrt;
scope_object_vector = getscope(tg, [1,2])

scope_object_vector =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 1
   Status               = Interrupted
   Type                 = Target
   NumSamples           = 500
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 5  : Signal Generator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 1
   TriggerSample        = 0
   DisplayMode          = Redraw (Graphical)
   YLimit               = Auto
   Grid                 = on
   Signals              = 5  : Signal Generator
                          6  : Sum

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 2
   Status               = Interrupted
   Type                 = Target
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 0  : Gain
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 2
   TriggerSample        = 0
   DisplayMode          = Redraw (Graphical)
   YLimit               = Auto
   Grid                 = on
   Signals              = 0  : Gain
                          1  : Gain1
```

```
                    2  : Gain2
```

- “Application and Driver Scripts”

Input Arguments

`target_object` — Object representing target computer

object variable

Object of type `SimulinkRealTime.target` that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required the Ethernet link settings.

Example: `tg`

Data Types: `struct`

`scope_number` — New scope number

`unsigned integer`

New scope number. This argument is optional. The default value is the next available integer in the target object property `Scopes`.

If you enter the scope number for an existing scope object, the result is an error.

Example: `1`

`scope_number_vector` — Vector of new scope numbers

`unsigned integer vector`

Vector of new scope numbers. If you enter the scope number for an existing scope object, the result is an error.

Example: `[2, 3]`

Output Arguments

`scope_object` — Object representing an existing scope

object

Object representing an existing scope

`scope_object_vector` — Vector of objects representing an existing scope
object

Vector containing objects representing an existing scope

See Also

Using Real-Time Application Objects | Real-Time Application Properties | `SimulinkRealTime.target.addscope` | `SimulinkRealTime.target.remscope`

Introduced in R2014a

SimulinkRealTime.target.getsignal

Value of signal

Syntax

```
signal_value = getsignal(target_object, signal_index)
```

Description

`signal_value = getsignal(target_object, signal_index)` returns the value of the signal associated with `signal_index` at the time the request is made. The value is not time-stamped. Successive calls to this function will not necessarily return successive signal values.

Examples

Get value of named signal

Get the value of signal `'Gain1'`.

Get signal index of `'Gain1'`

```
tg = slrt;
sid = getsignalid(tg, 'Gain1');
```

Get value of signal

```
getsignal(tg, sid)
```

```
ans =

-3.3869e+006
```

Input Arguments

`target_object` — Object representing target computer
object variable

Object of type `SimulinkRealTime.target` that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required the Ethernet link settings.

Example: `tg`

Data Types: `struct`

`signal_index` — Index number of the signal

nonnegative integer

To be accessible via signal index, the signal must be observable.

Example: `0`

Example: `1`

Output Arguments

`signal_value` — Value of signal

number | string

Virtual and bus signals, optimized signals, and signals of complex data types are not observable.

More About

- "Nonobservable Signals and Parameters"

See Also

Using Real-Time Application Objects | Real-Time Application Properties | `SimulinkRealTime.target.getsignalid`

Introduced in R2014a

SimulinkRealTime.target.getsignalid

Signal index from signal hierarchical name

Syntax

```
signal_index = getsignalid(target_object, signal_name)
```

Description

`signal_index = getsignalid(target_object, signal_name)` returns the index of a signal from the signal list, based on the path to the signal name.

You must enter the names in full. The names are case sensitive.

For block names, enter the mangled name that Simulink Coder uses for code generation.

Examples

Top-level block with single output

Get signal index for single output of block `Gain1`.

```
tg = slrt;
getsignalid(tg, 'Gain1')
```

```
ans =

6
```

Lower-level block with single output

Get signal index for single output of block `Feedback/Gain1`.

```
tg = slrt;
getsignalid(tg, 'Feedback/Gain1')
```

```
ans =

6
```

Top-level block with multiple outputs

Get signal index for output signal 2 of block `Byte Packing`.

```
tg = slrt;
signal_index = getsignalid(tg,'Byte Packing /s2')
```

```
signal_index =

           1
```

Input Arguments

`target_object` — Object representing target computer

object variable

Object of type `SimulinkRealTime.target` that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required the Ethernet link settings.

Example: `tg`

Data Types: `struct`

`signal_name` — Hierarchical name of signal from model

string

Simulink Real-Time constructs signal names in two ways:

- For blocks with a single signal, `signal_name` is the same as the block name.
- For blocks with multiple signals, Simulink Real-Time constructs `signal_name` by appending `' /s1'`, `' /s2'`, . . ., to the block name.

Example: `'Gain2'`

Example: `'Feedback/Gain1'`

Example: `'Byte Packing /s2'`

Output Arguments

`signal_index` — Index number of the signal

nonnegative integer

To be accessible via signal index, the signal must be observable.

Example: 0

Example: 1

More About

- “Nonobservable Signals and Parameters”

See Also

Using Real-Time Application Objects | Real-Time Application Properties | `SimulinkRealTime.target.getsignal`

Introduced in R2014a

SimulinkRealTime.target.getsignalidsfromlabel

Vector of signal indices

Syntax

```
index_vector = getsignalidsfromlabel(target_object, signal_label)
```

Arguments

`target_object`	Name of a target object.
`signal_label`	Signal label (from Simulink model).

Description

`index_vector = getsignalidsfromlabel(target_object, signal_label)` returns a vector of one or more signal indices that are associated with the labeled signal, `signal_label`.

You must have labeled the signal for which you request the index using the Simulink **Signal name** parameter. You must have applied a unique label. That is, only one signal has the label `signal_label`.

The Simulink Real-Time software refers to Simulink signal names as signal labels.

Examples

Get the vector of signal indices for a signal labeled `Gain`:

```
tg = slrt;
getsignalidsfromlabel(tg, 'xpcoscGain')
ans =

0
```

More About

- “Signal Properties Controls”

See Also

`SimulinkRealTime.target.getsignallabel`

Introduced in R2014a

SimulinkRealTime.target.getsignallabel

Signal label for signal index

Syntax

```
signal_label = getsignallabel(target_object, signal_index)
```

Arguments

`target_object`	Name of a target object.
`signal_index`	Index number of the signal.

Description

`signal_label = getsignallabel(target_object, signal_index)` returns the signal label for the specified signal index, `signal_index`.

You must have labeled the signal for which you request the index using the Simulink **Signal name** parameter. The Simulink Real-Time software refers to Simulink signal names as signal labels.

Examples

Get the signal label for signal index `0`:

```
tg = slrt;
getsignallabel(tg, 0)
ans =

xpcoscGain
```

More About

- “Signal Properties Controls”

See Also

`SimulinkRealTime.target.getsignalidsfromlabel`

Introduced in R2014a

SimulinkRealTime.target.getsignalname

Signal name from index list

Syntax

```
signal_name = getsignalname(target_object, signal_index)
```

Arguments

`target_object`	Name of a target object.
`signal_index`	Index number of the signal.
`signal_name`	Output name string of the signal.

Description

`signal_name = getsignalname(target_object, signal_index)` returns one string from the index list for the specified signal index.

The signal name refers to the block path of the block whose output is the specified signal. The software consru8cts the name according to the following rules:

- If the block in question has more than one output port, `'/pn'` is appended to the signal name, where `n` is the port number (starting at `1`).
- If the output port in question is not a scalar, `'/sn'` is appended to the signal name, where `n` is the index of signal `signal_index` within the vector or matrix. For this purpose, the signals are flattened to one dimension. For example, a `2 x 2` matrix will be represented by signals `/s1`, `/s2`, `/s3`, `/s4`.

These rules result in the following function behavior for block `Subsystem/path/to/block`:

- If the block has only one output port and the port is a scalar port, the function returns `Subsystem/path/to/block`.

- If the block has one output port, the port is a vector port, and `signal_index` refers to the second element within that vector, the function returns `Subsystem/path/to/block/s2`.
- If the block has three output ports, the second output port outputs a vector, and `signal_index` refers to the seventh element within that vector, the function returns `Subsystem/path/to/block/p2/s7`.
- If the block has three output ports, the second port outputs a scalar, and `signal_index` refers to the output from the second port, the function returns `Subsystem/path/to/block/p2`.

Examples

Get the signal name of signal index `2`:

```
tg = slrt;
sigName = getsignalname(tg,2)
sigName =
Gain2
```

Introduced in R2014a

SimulinkRealTime.target.load

Download real-time application to target computer

Syntax

```
target_object = load(target_object,real_time_application)
```

Description

`target_object = load(target_object,real_time_application)` loads the application real_time_application onto the target computer represented by target_object.

The call returns target_object, updated with the new state of the target.

Examples

Load `xpcosc`

Load the real-time application `xpcosc` into target computer `TargetPC1`, represented by target object `tg`. Start the application.

Get the target object.

```
tg = SimulinkRealTime.target('TargetPC1')
```

```
Simulink Real-Time Object

   Connected             = Yes
   Application           = loader
```

Load the real-time application.

```
load(tg, 'xpcosc')
```

```
Simulink Real-Time Object

   Connected             = Yes
   Application          = xpcosc
   Mode                 = Real-Time Single-Tasking
   Status               = stopped
   CPUOverload          = none

   ExecTime             = 0.0000
   SessionTime          = 918.5713
   StopTime             = 0.200000
   SampleTime           = 0.000250
   AvgTET               = NaN
   MinTET               = 9999999.000000
   MaxTET               = 0.000000
   ViewMode             = 0

   TimeLog              = Vector(0)
   StateLog             = Matrix (0 x 2)
   OutputLog            = Matrix (0 x 2)
   TETLog               = Vector(0)
   MaxLogSamples        = 16666
   NumLogWraps          = 0
   LogMode              = Normal

   Scopes               = No Scopes defined
   NumSignals           = 7
   ShowSignals          = off

   NumParameters        = 7
   ShowParameters       = off
```

Start the application.

```
start(tg)
```

- “Application and Driver Scripts”

Input Arguments

`target_object` — Object representing target computer

object variable

Object of type `SimulinkRealTime.target` that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required the Ethernet link settings.

Example: `tg`

Data Types: `struct`

`real_time_application` — Name of real-time application

string

Name of the real-time application, without file extension. real_time_application can also contain the absolute path to the real-time application, without file extension.

You must build the application in the working folder on the development computer. By default, after the Simulink Coder build process is complete, the Simulink Real-Time software calls `SimulinkRealTime.target.load`. If a real-time application was previously loaded, before downloading the new real-time application, `SimulinkRealTime.target.load` unloads the old real-time application.

If you are running the real-time application in Standalone mode, a call to `SimulinkRealTime.target.load` has no effect. To load a new application, rebuild the standalone application files with the new application and transfer the updated files to the target computer using `SimulinkRealTime.fileSystem`. Then, restart the target computer with the new standalone application.

Data Types: `char`

See Also

Using Real-Time Application Objects | Real-Time Application Properties | `SimulinkRealTime.target.unload`

Introduced in R2014a

SimulinkRealTime.target.loadparamset

Restore parameter values saved in specified file

Syntax

```
loadparamset(target_object,'filename')
```

Arguments

`target_object`	Name of an existing target object.
`filename`	Enter the name of the file that contains the saved parameters.

Description

`loadparamset(target_object,'filename')` restores the real-time application parameter values saved in the file `filename`. Save this file on a local drive of the target computer. You must have a parameter file from a previous run of the `SimulinkRealTime.target.saveparamset` method.

See Also

`SimulinkRealTime.target.saveparamset`

Introduced in R2014a

SimulinkRealTime.target.ping

Test communication between development and target computers

Syntax

```
status_value = ping(target_object)
```

Description

`status_value = ping(target_object)` tests whether the development computer and the target computer represented by `target_object` can communicate using the settings stored in `target_object`.

Examples

Check communication with default target computer

```
target_object = slrt;
ping(target_object)
```

```
ans =

success
```

Check communication with specified target computer, not started

```
target_object = slrt('TargetPC1');
ping(target_object)
```

```
ans =

failed
```

Input Arguments

`target_object` — Object representing target computer

object variable

Object of type SimulinkRealTime.target that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required the Ethernet link settings.

Example: tg

Data Types: struct

Output Arguments

status_value — Reports if the kernel is loaded and communication is working
'success' | 'failed'

Simulink Real-Time kernel is loaded and running, and communication is working between the development and target computers.

See Also

Using Real-Time Application Objects | Real-Time Application Properties | SimulinkRealTime.target.reboot

Introduced in R2014a

SimulinkRealTime.target.reboot

Restart target computer

Syntax

```
reboot(target_object)
```

Description

`reboot(target_object)` restarts the target computer. If a target boot disk is still present, `reboot` reloads the Simulink Real-Time kernel.

At the target computer command line, you can use the corresponding command:

```
reboot
```

Examples

Reboot target computer `'TargetPC1'`

Get a target object and reboot the computer that it represents

Get target object for target computer `'TargetPC1'`

```
tg = SimulinkRealTime.target('TargetPC1')
```

```
Target: TargetPC1
   Connected            = Yes
   Application          = loader
```

Restart target computer.

```
reboot(tg)
```

Input Arguments

`target_object` — Object representing target computer
object variable

Object of type SimulinkRealTime.target that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required the Ethernet link settings.

Example: tg

Data Types: struct

See Also

"Target Computer Commands" | Using Real-Time Application Objects | Real-Time Application Properties | SimulinkRealTime.target.load | SimulinkRealTime.target.unload

Introduced in R2014a

SimulinkRealTime.target.remscope

Remove scope from target computer

Syntax

```
remscope(target_object)
remscope(target_object, scope_number)
remscope(target_object, scope_number_vector)
```

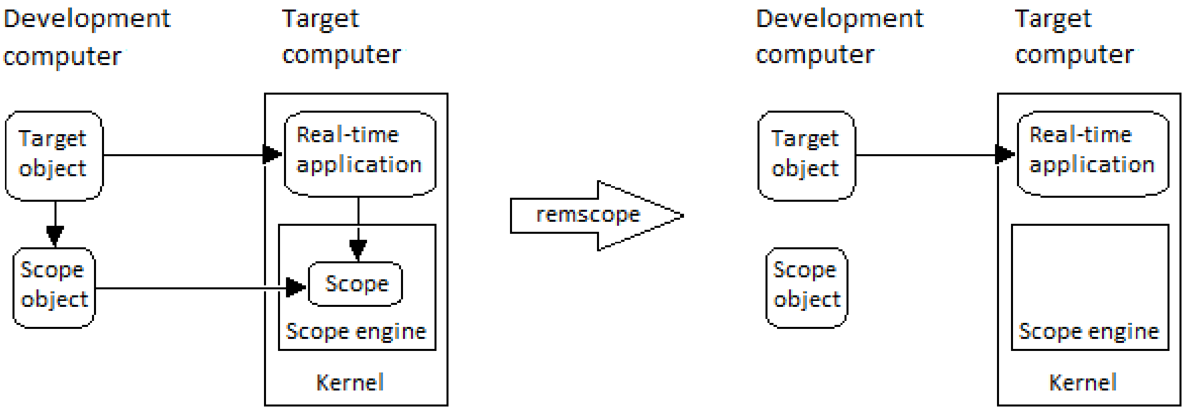
Description

`remscope(target_object)` deletes all scopes from the target computer.

`remscope(target_object, scope_number)` deletes the scope represented by `scope_number` from the target computer.

`remscope(target_object, scope_number_vector)` deletes the scopes represented by the scope numbers listed in `scope_number_vector` from the target computer.

The method `remscope` has no return value. `remscope` does not delete the scope object that represents the scope on the development computer.



You can permanently remove only a scope that is added with the method addscope. This scope is outside the model. If you remove a scope that a scope block added inside the model, a subsequent run of that model recreates the scope.

At the target computer command line, you can remove one scope or all scopes:

```
remscope scope_number
remscope all
```

Examples

Remove all scopes

```
tg = slrt;
remscope(tg)
```

Remove one scope

```
tg = slrt;
remscope(tg,1)
```

Remove vector of two scopes

```
tg = slrt;
remscope(tg,[1 2])
```

Input Arguments

target_object — Object representing target computer

object variable

Object of type SimulinkRealTime.target that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required the Ethernet link settings.

Example: tg

Data Types: struct

scope_number — New scope number

unsigned integer

New scope number. This argument is optional. The default value is the next available integer in the target object property `Scopes`.

If you enter the scope number for an existing scope object, the result is an error.

Example: `1`

`scope_number_vector` — Vector of new scope numbers
`unsigned integer vector`

Vector of new scope numbers. If you enter the scope number for an existing scope object, the result is an error.

Example: `[2, 3]`

See Also

“Target Computer Commands” | Using Real-Time Application Objects | Real-Time Application Properties | `SimulinkRealTime.target.addscope` | `SimulinkRealTime.target.getscope`

Introduced in R2014a

SimulinkRealTime.target.saveparamset

Save real-time application parameter values

Syntax

```
saveparamset(target_object,'filename')
```

Arguments

`target_object`	Name of an existing target object.
`filename`	File name to contain the saved parameters.

Description

`saveparamset(target_object,'filename')` saves the real-time application parameter values in the file `filename`. This method saves the file on a local drive of the target computer (C:\ by default). You can later reload these parameters with the `loadparamset` function.

Save real-time 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 real-time application parameter values from a number of application runs.

See Also

`SimulinkRealTime.target.loadparamset`

Introduced in R2014a

SimulinkRealTime.target.setparam

Change value of tunable parameter in real-time application

Syntax

```
[parIndexVec,OldValues,NewValues] = setparam(target_object,
parameter_index, parameter_value)
[parIndexVec,OldValues,NewValues] = setparam(target_object,
parameter_index_vec, parameter_value_vec)
```

Description

`[parIndexVec,OldValues,NewValues] = setparam(target_object, parameter_index, parameter_value)` sets the value of the target parameter, indicated by a parameter index or a vector of parameter indexes, to a new value. If the second argument is a vector, the third argument must be a cell array. This method returns a structure that stores the parameter index, previous parameter values, and new parameter values.

`[parIndexVec,OldValues,NewValues] = setparam(target_object, parameter_index_vec, parameter_value_vec)` sets the value of the target parameter, indicated by a parameter index or a vector of parameter indexes, to a new value. If the second argument is a vector, the third argument must be a cell array. This method returns a structure that stores the parameter index, previous parameter values, and new parameter values.

Examples

Set block parameter value given names of parameter and block

Set the value of block parameter `'Gain'` of block `'Gain1'` to `10000000`.

Get block parameter index from parameter hierarchical name

```
tg = slrt;
pid = getparamid(tg, 'Gain1', 'Gain');
```

Set value of block parameter

```
setparam(tg, pid, 10000000)
```

```
ans =

    parIndexVec: 1
      OldValues: 6000
      NewValues: 10000000
```

Set model parameter value given name of parameter

Set the value of model parameter `'G2'` to `10000000`.

Get model parameter index from parameter name

```
tg = slrt;
pid = getparamid(tg, '', 'G2');
```

Set value of model parameter

```
setparam(tg, pid,10000000)
```

```
ans =

    parIndexVec: 4
      OldValues: 100
      NewValues: 10000000
```

Sweep block parameter value

Sweep the value of block parameter `'Gain'` of block `'Gain1'` by steps of `2000`.

Get block parameter index from parameter hierarchical name

```
tg = slrt;
pid = getparamid(tg, 'Gain1', 'Gain');
```

Set value of block parameter

```
for i = 1 : 3
    setparam(tg, pid, (i*2000))
end
```

```
ans =
    parIndexVec: 1
      OldValues: 10000000
      NewValues: 2000


ans = 

    parIndexVec: 1
      OldValues: 2000
      NewValues: 4000


ans = 

    parIndexVec: 1
      OldValues: 4000
      NewValues: 6000
```

Simultaneously set block parameter values for multiple parameters

Set the value of block parameter `'Gain'` of blocks `'Gain1'` and `'Gain2'` to `10000000` and `400` respectively.

Get block parameter index from parameter hierarchical name

```
tg = slrt;
pid1 = getparamid(tg, 'Gain1', 'Gain');
pid2 = getparamid(tg, 'Gain2', 'Gain');
```

Set new values of block parameters

```
setparam(tg, [pid1, pid2], {10000000, 400})
```

```
ans = 

    parIndexVec: [1 2]
      OldValues: {[400]  [10000000]}
      NewValues: {[10000000]  [400]}
```

Input Arguments

`target_object` — Object representing target computer

object variable

Object of type `SimulinkRealTime.target` that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required the Ethernet link settings.

Example: `tg`

Data Types: `struct`

`parameter_index` — Index number of the parameter
nonnegative integer

Parameter index returned by `SimulinkRealTime.target.getparamid`

Example: 1

`parameter_value` — New parameter value
number | string

New value with data type as required by parameter

Example: 1

`parameter_index_vec` — Vector of parameter index numbers
vector of nonnegative integers

Parameter indexes returned by `SimulinkRealTime.target.getparamid`

Example: `[1,2,3]`

`parameter_value_vec` — Cell array of new parameter values
cell array of numbers and strings

New values with data types as required by parameter

Example: `{1,2,3}`

Output Arguments

`parIndexVec` — Parameter index of changed parameter or vector of parameter indexes
nonnegative integer | vector of nonnegative integers

Structure element named `parIndexVec` containing the parameter index or indexes passed in using the `parameter_index_vec` argument.

Example: `1`

Example: `[1,5]`

`OldValues` — Values held by parameter or parameters before change

number | string | cell array of numbers and strings

Structure element named `OldValues` containing the new value (number or string) or a cell array containing individual numbers or strings, packaged as vectors.

Example: `OldValues: 10`

Example: `OldValues: {[400] [2]}`

`NewValues` — Values held by parameter or parameters after change

number | string | cell array of numbers and strings

Structure element named `NewValues` containing the new value (number or string) or a cell array containing individual numbers or strings, packaged as vectors.

Example: `New Values: 400`

Example: `NewValues: {[10] [100]}`

More About

- “Parameter Tuning Basics”
- “Nonobservable Signals and Parameters”

See Also

Using Real-Time Application Objects | Real-Time Application Properties | `SimulinkRealTime.target.getparam` | `SimulinkRealTime.target.getparamid`

Introduced in R2014a

SimulinkRealTime.target.start

Start execution of real-time application on target computer

Syntax

```
start(target_object)
```

Description

`start(target_object)` starts execution of the real-time application represented by the target object. Before using this method, you must create and load the real-time application on the target computer. If a real-time application is running, this command has no effect.

At the target computer command line, you can use the corresponding command:

```
start
```

Examples

Start real-time application `tg`

Start the real-time application represented by the target object `tg`

```
tg = slrt;
start(tg)
```

Input Arguments

`target_object` — Object representing target computer
object variable

Object of type `SimulinkRealTime.target` that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required the Ethernet link settings.

Example: `tg`

Data Types: `struct`

See Also

“Target Computer Commands” | Using Real-Time Application Objects | Real-Time Application Properties | `SimulinkRealTime.target.stop`

Introduced in R2014a

SimulinkRealTime.target.stop

Stop execution of real-time application on target computer

Syntax

```
stop(target_object)
```

Description

`stop(target_object)` stops execution of the real-time application represented by the target object. Before using this method, you must create and load the real-time application on the target computer. If a real-time application is not running, this command has no effect.

At the target computer command line, you can use the corresponding command:

```
stop
```

Examples

Stop real-time application `tg`

Stop the real-time application represented by the target object `tg`

```
tg = slrt;
stop(tg)
```

Input Arguments

`target_object` — Object representing target computer
object variable

Object of type `SimulinkRealTime.target` that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required the Ethernet link settings.

Example: `tg`

Data Types: `struct`

See Also

"Target Computer Commands" | Using Real-Time Application Objects | Real-Time Application Properties | `SimulinkRealTime.target.start`

Introduced in R2014a

SimulinkRealTime.target.unload

Remove real-time application from target computer

Syntax

```
unload(target_object)
```

Description

`unload(target_object)` removes the loaded real-time application from the target computer. The kernel goes into loader mode and is ready to download new real-time application from the development computer.

If you are running the real-time application in `Stand Alone` mode, this command has no effect. To unload and reload a new application, you must rebuild the standalone application with the new application, and then restart the target computer with the updated standalone application.

Examples

Unload real-time application

Unload the real-time application represented by the target object `tg`.

Unload the real-time application.

```
tg = slrt;
unload(tg);
```

```
Target: TargetPC1
   Connected            = Yes
   Application          = loader
```

Input Arguments

`target_object` — Object representing target computer
object variable

Object of type `SimulinkRealTime.target` that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required the Ethernet link settings.

Example: `tg`

Data Types: `struct`

See Also

Using Real-Time Application Objects | Real-Time Application Properties | `SimulinkRealTime.target.load` | `SimulinkRealTime.target.reboot`

Introduced in R2014a

SimulinkRealTime.target.viewTargetScreen

Open real-time window on development computer

Syntax

```
viewTargetScreen(target_object)
```

Description

`viewTargetScreen(target_object)` opens a Simulink Real-Time display window for `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 development computer screen with another target screen, move the cursor into the display window, right-click, and select **Update Simulink Real-Time Target Screen**.
- If `TargetScope` is disabled, text output is transferred once every second to the development computer and displayed in the window.

Examples

View screen for default target computer

Get target object for default computer, open window display with target computer screen

```
tg = slrt;
viewTargetScreen(tg)
```

View screen for target computer `'TargetPC1'`

Get target object for `'TargetPC1'`, open window display with target computer screen

```
tg = slrt('TargetPC1');
viewTargetScreen(tg)
```

Input Arguments

`target_object` — Object representing target computer

object variable

Object of type `SimulinkRealTime.target` that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required the Ethernet link settings.

Example: `tg`

Data Types: `struct`

SimulinkRealTime.fileScope

Control and access properties of file scopes

Description

The scope gets a data package from the kernel and stores the data in a file on the target computer file system. Depending on the setting of `WriteMode`, the file size is or is not continuously updated. You can transfer the data to another computer for examination or plotting.

Methods

The methods in the following table apply to file, host, and target scopes.

Method	Description
`SimulinkRealTime.fileSc`	Add signals to scope represented by scope object
`SimulinkRealTime.fileSc`	Remove signals from scope represented by scope object
`SimulinkRealTime.fileSc`	Start execution of scope on target computer
`SimulinkRealTime.fileSc`	Stop execution of scope on target computer
`SimulinkRealTime.fileSc`	Software trigger start of data acquisition for scope or scopes

Properties

Scope object properties let you select signals to acquire, set triggering modes, and access signal information from the real-time application.

To get the value of a readable scope object property from a scope object:

```
scope_object = getscope(target_object, scope_number);
value = scope_object.scope_object_property
```

For example, to get the `Decimation` of scope `3`:

```
scope_object = getscope(tg, 3);
value = scope_object.Decimation
```

To set the value of a writable scope property from a scope object:

```
scope_object = getscope(target_object, scope_number);
scope_object.scope_object_property = new_value
```

For example, to set the `Decimation` of scope `3`:

```
scope_object = getscope(tg, 3);
scope_object.Decimation = 10
```

Not all properties are user-writable. For example, property `Type` is not writable after you have created the scope.

The properties in the following table apply to file, host, and target scopes.

Property	Description	Writable
`Application`	Name of the Simulink model associated with this scope object.	No
`Decimation`	A number `n`; every `n`th sample is acquired by a scope.	Yes
`NumPrePostSamples`	Number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set `TriggerMode` to `'FreeRun'`, this property has no effect on data acquisition.	Yes
`NumSamples`	Number of contiguous samples captured during the acquisition of a data package. If the scope stops before capturing this number of samples, the scope has the collected data up to the end of data collection, then has zeroes for the remaining uncollected data. Know what type of data you are collecting, because it is possible that your data contains zeroes. For file scopes, this parameter works in conjunction with the **AutoRestart** check box. If you select the **AutoRestart** box, the file scope collects data up to **Number of Samples**, then starts over again, overwriting the buffer. If you do not select the **AutoRestart** box, the file scope collects data only up to **Number of Samples**, then stops.	Yes
`ScopeId`	A numeric index, unique for each scope.	No

Property	Description	Writable
`Signals`	List of signal indices from the target object to display on the scope.	Yes
`Status`	Indicates whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are `'Acquiring'`, `'Ready for being Triggered'`, `'Interrupted'`, and `'Finished'`.	No
`TriggerLevel`	If `TriggerMode` is `'Signal'`, indicates the value the signal has to cross to trigger the scope to start acquiring data. The trigger level can be crossed with a rising or a falling signal.	Yes
`TriggerMode`	Trigger mode for a scope. Valid values are: • `'freerun'` — scope triggers on every sample time. • `'software'` — scope triggers from Command Window. • `'signal'` — scope triggers when a designated signal changes state. • `'scope'` — scope triggers when a designated scope triggers. The default value is `'FreeRun'`.	Yes

Property	Description	Writable
`TriggerSample`	If `TriggerMode` is `'Scope'`, then `TriggerSample` specifies which sample of the triggering scope to trigger on. For example, if `TriggerSample` is `0` (default), the current scope triggers on sample `0` (first sample acquired) of the triggering scope. 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. Setting `TriggerSample` to `-1` means that the current scope is triggered at the end of the acquisition cycle of the triggering scope. The first sample of the triggering scope is acquired one sample after the last sample of the triggering scope.	Yes
`TriggerScope`	If `TriggerMode` is `'Scope'`, it identifies the scope to use for a trigger. You can set a scope to trigger when another scope is triggered. Set the slave scope property `TriggerScope` to the scope index of the master scope.	Yes
`TriggerSignal`	If `TriggerMode` is `'Signal'`, it identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property `Signal`.	Yes
`TriggerSlope`	If `TriggerMode` is `'Signal'`, it indicates whether the trigger is on a rising signal, falling signal, or either. • `either` — the signal triggers the scope when it crosses `triggerlevel` in either the rising or falling directions. • `rising` — the signal triggers the scope when it crosses `triggerlevel` in the rising direction. • `falling` — the signal triggers the scope when it crosses `triggerlevel` in the falling direction. The default value is `'Either'`.	Yes

Property	Description	Writable
`Type`	Determines how the development computer collects and displays its data. • `'Host'` — the data is collected on the target computer and displayed on the development computer. • `'Target'` — the data is collected on the target computer and displayed on the target computer monitor. • `'File'` — the data is collected and stored on the target computer. You set property `Type` only once, when you create the scope on the target computer.	No

The properties in the following table apply only to file scopes.

Property	Description	Writeable
`AutoRestart`	Values are `'on'` and `'off'`. For file scopes, enables the file scope to collect data up to the number of samples (`NumSamples`), then start over again, appending the new data to the end of the signal data file. Clear the **AutoRestart** check box to have the file scope collect data up to **Number of samples**, then stop. When you start the real-time application, if the named signal data file already exists, the software overwrites the old data with the new signal data. Set `AutoRestart` to `'on'` before enabling `DynamicFileName`.	No

Property	Description	Writeable
	For host or target scopes, this parameter has no effect.	
`DynamicFileNar`	Values are `'on'` and `'off'`. By default, the value is `'off'`. Enables the ability to dynamically create multiple log files for file scopes. Set `AutoRestart` to `'on'` before enabling `DynamicFileName`. Configure `Filename` to create incrementally numbered file names for the multiple log files. If you do not do this, the software generates an error when you try to start the scope. You can enable the creation of up to 99999999 files (`<%%%%%%%%>.dat`). The length of a file name, including the specifier, cannot exceed eight characters. For host or target scopes, this parameter has no effect.	Yes

Property	Description	Writeable

`Filename`	Provide a name for the file to contain the signal data. By default, the target computer writes the signal data to a file named `C:\data.dat` for scope blocks. For file scopes that you create through the MATLAB interface, no name is initially assigned to `FileName`. After you start the scope, the software assigns a name for the file to acquire the signal data. This name typically consists of the scope object name, `ScopeId`, and the beginning letters of the first signal added to the scope.
	If you set `DynamicFileName` and `AutoRestart` to `'on'`, configure `Filename` to dynamically increment. Use a base file name, an underscore (_), and a `< >` specifier. Within the specifier, enter one to eight `%` symbols. Each symbol `%` represents a decimal location in the file name. The specifier can appear anywhere in the file name. For example, the following value for `Filename`, `C:\work\file_<%%%>.dat` creates file names with the following pattern:
	`file_001.dat`
	`file_002.dat`
	`file_003.dat`
	The last file name of this series is `file_999.dat`. If the function is still logging data when the last file name reaches its maximum size, the function starts from the beginning and overwrites the first file name in the series. If you do not retrieve the
		No

Property	Description	Writeable
	data from existing files before they are overwritten, the data is lost. For host or target scopes, this parameter has no effect.	
`MaxWriteFileSi`	Provide the maximum size of `Filename`, in bytes. This value must be a multiple of `WriteSize`. Default is `536870912`. When the size of a log file reaches `MaxWriteFileSize`, the software creates a subsequently numbered file name. The software continues logging data to that file, up until the highest log file number that you specified. If the software cannot create additional log files, it overwrites the first log file. For host or target scopes, this parameter has no effect.	Yes
`Mode`	**Note:** The `Mode` property will be removed in a future release. • For target scopes, use `DisplayMode`. • For file scopes, use `WriteMode`. • For host scopes, this parameter has no effect.	Yes

Property	Description	Writeable
`WriteMode`	For file scopes, specify when a file allocation table (FAT) entry is updated. Values are `'Lazy'` or `'Commit'`. Both modes write the signal data to the file. With `'Commit'` mode, each file write operation simultaneously updates the FAT entry for the file. This mode is slower, but the file system maintains the actual file size. With `'Lazy'` mode, the FAT entry is updated only when the file is closed. It is not updated during each file write operation. This mode is faster, but if the system stops responding before the file is closed, the file system might not know the actual file size (the file contents, however, are intact). For host or target scopes, this parameter has no effect.	Yes
`WriteSize`	Enter the block size, in bytes, of the data chunks. This parameter specifies that a memory buffer, of length number of samples (`NumSamples`), collect data in multiples of `WriteSize`. By default, this parameter is 512 bytes, which is the typical disk sector size. Using a block size that is the same as the disk sector size provides better performance. If your system stops responding, you can expect to lose an amount of data the size of `WriteSize`. For host or target scopes, this parameter has no effect.	Yes

More About

- “File Scope Usage”

Introduced in R2014a

SimulinkRealTime.fileScope.addsignal

Add signals to file scope represented by scope object

Syntax

```
addsignal(scope_object_vector, signal_index_vector)
```

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.

Description

`addsignal(scope_object_vector, signal_index_vector)` adds signals to a scope object. The signals must be specified by their indices, which you can retrieve using the target object method `SimulinkRealTime.target.getsignalid`. If `scope_object_vector` has two or more scope objects, the same signals are assigned to each scope.

Before you can add a signal to a scope, you must stop the scope.

At the target computer command line, you can add multiple signals to the scope:

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

Examples

The following examples use model `xpcosc`.

Add signals `Integrator1` and `Signal Generator` to scope object `sc1`.

```
tg = slrt;
sc1 = addscope(tg,'file',1);
s0 = getsignalid(tg,'Signal Generator');
s1 = getsignalid(tg,'Integrator1');
addsignal(sc1,[s0,s1]);
```

The scope object property `Signals` is updated to include the added signals. Type `sc1` to display the properties and values for scope `sc1`.

More About

- “File Scope Usage”

See Also

“Target Computer Commands” | `SimulinkRealTime.fileScope.remsignal` | `SimulinkRealTime.target.addscope` | `SimulinkRealTime.target.getsignalid`

Introduced in R2014a

SimulinkRealTime.fileScope.remsignal

Remove signals from file scope represented by scope object

Syntax

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

Arguments

`scope_object_vector` Scope object or vector of scope objects. The target object methods `addscope` or `getscope` create scope objects.

`signal_index_vector` Index numbers from the scope object property `Signals`. This argument is optional. If it is left out, all signals are removed.

Description

`remsignal(scope_object)` removes all signals from a scope object.

`remsignal(scope_object, signal_index_vector)` 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 `scope_object` is a vector containing two or more scope objects, the same signals are removed from each scope.

Before you can remove a signal from a scope, you must stop the scope.

At the target computer command line, you can remove multiple signals from the scope:

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

`signal_index` is optional. If it is left out, all signals are removed.

Examples

The following examples use model `xpcosc`.

Remove all signals from the scope represented by the scope object sc1:

```
tg = slrt;
sc1 = getscope(tg,1);
remsignal(sc1)
```

Remove signals `Integrator1` and `Signal Generator` from the scope on the target computer:

```
tg = slrt;
sc1 = getscope(tg,1);
s0 = getsignalid(tg,'Signal Generator');
s1 = getsignalid(tg,'Integrator1');
remsignal(sc1,[s0,s1])
```

More About

- "File Scope Usage"

See Also

"Target Computer Commands" | `SimulinkRealTime.fileScope.addsignal` | `SimulinkRealTime.target.getsignalid` | `SimulinkRealTime.target.remscope`

Introduced in R2014a

SimulinkRealTime.fileScope.start

Start execution of file scope on target computer

Syntax

```
start(scope_object)
start(scope_object_vector)
start([scope_object1, scope_object2, . . ., scope_objectN])
```

Arguments

`scope_object`	Name of single vector object.
`scope_object_vector`	Name of vector of scope objects.

Description

`start(scope_object)` starts a scope on the target computer represented by a scope object on the development computer. This method might not start data acquisition, which depends on the trigger settings.

Before using this method, you must create a scope. To create a scope, use the target object method `addscope` or add Simulink Real-Time scope blocks to your Simulink model.

Alternative syntaxes are `start(scope_object_vector)`and `start([scope_object1, scope_object2, . . ., scope_objectN])`.

At the target computer command line, you can use the corresponding command:

```
startscope scope_index
startscope all
```

Examples

Start one scope with the scope object `sc1`:

```
tg = slrt;
sc1 = getscope(tg,1)
start(sc1)
```

Start two scopes, 1 and 2:

```
tg = slrt;
somescopes = getscope(tg,[1,2])
start(somescopes)
```

or

```
tg = slrt;
sc1 = getscope(tg,1)
sc2 = getscope(tg,2)
start([sc1,sc2])
```

Start all scopes:

```
tg = slrt;
allscopes = getscope(tg)
start(allscopes)
```

More About

- "File Scope Usage"

See Also

"Target Computer Commands" | `SimulinkRealTime.fileScope.stop` | `SimulinkRealTime.target.getscope` | `SimulinkRealTime.target.start`

Introduced in R2014a

SimulinkRealTime.fileScope.stop

Stop execution of file scope on target computer

Syntax

```
stop(scope_object)
stop(scope_object_vector)
stop([scope_object1, scope_object2, . . ., scope_objectN])
```

Arguments

`scope_object`	Name of single vector object.
`scope_object_vector`	Name of vector of scope objects.

Description

`stop(scope_object)` stops a scope on the target computer represented by a scope object on the development computer.

Alternative syntaxes are `stop(scope_object_vector)` and `stop([scope_object1, scope_object2, . . ., scope_objectN])`.

At the target computer command line, you can use the corresponding command:

```
stopscope scope_index
stopscope all
```

Examples

Stop one scope with the scope object `sc1`:

```
tg = slrt;
sc1 = getscope(tg,1)
stop(sc1)
```

Stop two scopes, 1 and 2:

```
tg = slrt;
somescopes = getscope(tg,[1,2])
stop(somescopes)
```

or

```
tg = slrt;
sc1 = getscope(tg,1)
sc2 = getscope(tg,2)
stop([sc1,sc2])
```

Stop all scopes:

```
tg = slrt;
allscopes = getscope(tg)
stop(allscopes)
```

More About

- “File Scope Usage”

See Also

“Target Computer Commands” | `SimulinkRealTime.fileScope.start` | `SimulinkRealTime.target.getscope` | `SimulinkRealTime.target.stop`

Introduced in R2014a

SimulinkRealTime.fileScope.trigger

Software-trigger start of data acquisition for file scope

Syntax

```
trigger(scope_object_vector)
```

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

`trigger(scope_object_vector)` triggers the scope represented by the scope object to acquire the number of data points in the scope object property `NumSamples`.

If the scope object property `TriggerMode` has a value of `'Software'`, this function is the only way to trigger the scope. However, this function can be used on any scope, regardless of trigger mode setting. For example, if a scope is signal triggered and did not trigger because the triggering criteria were not met, this function can be used to force the scope to trigger.

Examples

Using model `xpcosc`, set a single file scope to software trigger, trigger the acquisition of one set of samples, read the file, and plot the data on the host.

```
tg = slrt;
tg.StopTime = Inf
sc1 = addscope(tg,'file',1);
```

```
sc1.FileName = 'data.dat';
addsignal(sc1, 4)
sc1.TriggerMode = 'software';
start(tg)
start(sc1)
trigger(sc1)
pause(0.5)
stop(sc1)
stop(tg)
fsys = SimulinkRealTime.fileSystem(tg);
hdl = fopen(fsys,'data.dat');
ddata = fread(fsys,hdl);
fclose(fsys, hdl);
mdata = SimulinkRealTime.utils.getFileScopeData(ddata);
plot(mdata.data(:,2),mdata.data(:,1))
```

More About

- “File Scope Usage”

Introduced in R2014a

SimulinkRealTime.hostScope

Control and access properties of host scopes

Description

The kernel acquires a data package and sends it to the scope. The scope waits for an upload command from the development computer and uploads the data. The development computer displays the data using a scope viewer or other MATLAB functions.

Methods

The methods in the following table apply to file, host, and target scopes.

Method	Description
`SimulinkRealTime.hostSc`	Add signals to scope represented by scope object
`SimulinkRealTime.hostSc`	Remove signals from scope represented by scope object
`SimulinkRealTime.hostSc`	Start execution of scope on target computer
`SimulinkRealTime.hostSc`	Stop execution of scope on target computer
`SimulinkRealTime.hostSc`	Software trigger start of data acquisition for scope or scopes

Properties

Scope object properties let you select signals to acquire, set triggering modes, and access signal information from the real-time application.

To get the value of a readable scope object property from a scope object:

```
scope_object = getscope(target_object, scope_number);
value = scope_object.scope_object_property
```

For example, to get the `Decimation` of scope `3`:

```
scope_object = getscope(tg, 3);
value = scope_object.Decimation
```

To set the value of a writable scope property from a scope object:

```
scope_object = getscope(target_object, scope_number);
scope_object.scope_object_property = new_value
```

For example, to set the `Decimation` of scope `3`:

```
scope_object = getscope(tg, 3);
scope_object.Decimation = 10
```

Not all properties are user-writable. For example, property `Type` is not writable after you have created the scope.

The properties in the following table apply to file, host, and target scopes.

Property	Description	Writable
`Application`	Name of the Simulink model associated with this scope object.	No
`Decimation`	A number `n`; every `n`th sample is acquired by a scope.	Yes
`NumPrePostSamples`	Number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set `TriggerMode` to `'FreeRun'`, this property has no effect on data acquisition.	Yes
`NumSamples`	Number of contiguous samples captured during the acquisition of a data package. If the scope stops before capturing this number of samples, the scope has the collected data up to the end of data collection, then has zeroes for the remaining uncollected data. Know what type of data you are collecting, because it is possible that your data contains zeroes. For file scopes, this parameter works in conjunction with the **AutoRestart** check box. If you select the **AutoRestart** box, the file scope collects data up to **Number of Samples**, then starts over again, overwriting the buffer. If you do not select the **AutoRestart** box, the file scope collects data only up to **Number of Samples**, then stops.	Yes
`ScopeId`	A numeric index, unique for each scope.	No

Property	Description	Writable
`Signals`	List of signal indices from the target object to display on the scope.	Yes
`Status`	Indicates whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are `'Acquiring'`, `'Ready for being Triggered'`, `'Interrupted'`, and `'Finished'`.	No
`TriggerLevel`	If `TriggerMode` is `'Signal'`, indicates the value the signal has to cross to trigger the scope to start acquiring data. The trigger level can be crossed with a rising or a falling signal.	Yes
`TriggerMode`	Trigger mode for a scope. Valid values are: • `'freerun'` — scope triggers on every sample time. • `'software'` — scope triggers from Command Window. • `'signal'` — scope triggers when a designated signal changes state. • `'scope'` — scope triggers when a designated scope triggers. The default value is `'FreeRun'`.	Yes

Property	Description	Writable
`TriggerSample`	If `TriggerMode` is `'Scope'`, then `TriggerSample` specifies which sample of the triggering scope to trigger on. For example, if `TriggerSample` is `0` (default), the current scope triggers on sample `0` (first sample acquired) of the triggering scope. 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. Setting `TriggerSample` to `-1` means that the current scope is triggered at the end of the acquisition cycle of the triggering scope. The first sample of the triggering scope is acquired one sample after the last sample of the triggering scope.	Yes
`TriggerScope`	If `TriggerMode` is `'Scope'`, it identifies the scope to use for a trigger. You can set a scope to trigger when another scope is triggered. Set the slave scope property `TriggerScope` to the scope index of the master scope.	Yes
`TriggerSignal`	If `TriggerMode` is `'Signal'`, it identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property `Signal`.	Yes
`TriggerSlope`	If `TriggerMode` is `'Signal'`, it indicates whether the trigger is on a rising signal, falling signal, or either. • `either` — the signal triggers the scope when it crosses `triggerlevel` in either the rising or falling directions. • `rising` — the signal triggers the scope when it crosses `triggerlevel` in the rising direction. • `falling` — the signal triggers the scope when it crosses `triggerlevel` in the falling direction. The default value is `'Either'`.	Yes

Property	Description	Writable
`Type`	Determines how the development computer collects and displays its data. • `'Host'` — the data is collected on the target computer and displayed on the development computer. • `'Target'` — the data is collected on the target computer and displayed on the target computer monitor. • `'File'` — the data is collected and stored on the target computer. You set property `Type` only once, when you create the scope on the target computer.	No

The properties in the following table apply only to host scopes.

Property	Description	Writeable
`Data`	Contains the output data for a single data package from a scope. For target or file scopes, this parameter has no effect.	No
`Time`	Contains the time data for a single data package from a scope. For target or file scopes, this parameter has no effect.	No

More About

- “Host Scope Usage”

Introduced in R2014a

SimulinkRealTime.hostScope.addsignal

Add signals to host scope represented by scope object

Syntax

```
addsignal(scope_object_vector, signal_index_vector)
```

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.

Description

`addsignal(scope_object_vector, signal_index_vector)` adds signals to a scope object. The signals must be specified by their indices, which you can retrieve using the target object method `SimulinkRealTime.target.getsignalid`. If `scope_object_vector` has two or more scope objects, the same signals are assigned to each scope.

Before you can add a signal to a scope, you must stop the scope.

At the target computer command line, you can add multiple signals to the scope:

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

Examples

The following examples use model `xpcosc`.

Add signals `Integrator1` and `Signal Generator` to scope object `sc1`.

```
tg = slrt;
sc1 = addscope(tg,'host',1);
s0 = getsignalid(tg,'Signal Generator');
s1 = getsignalid(tg,'Integrator1');
addsignal(sc1,[s0,s1]);
```

The scope object property `Signals` is updated to include the added signals. Type `sc1` to display the properties and values for scope `sc1`.

More About

- "Host Scope Usage"

See Also

"Target Computer Commands" | `SimulinkRealTime.hostScope.remsignal` | `SimulinkRealTime.target.addscope` | `SimulinkRealTime.target.getsignalid`

Introduced in R2014a

SimulinkRealTime.hostScope.remsignal

Remove signals from host scope represented by scope object

Syntax

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

Arguments

`scope_object_vector`	Scope object or vector of scope objects. The target object methods `addscope` or `getscope` create scope objects.
`signal_index_vector`	Index numbers from the scope object property `Signals`. This argument is optional. If it is left out, all signals are removed.

Description

`remsignal(scope_object)` removes all signals from a scope object.

`remsignal(scope_object, signal_index_vector)` 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 `scope_object` is a vector containing two or more scope objects, the same signals are removed from each scope.

Before you can remove a signal from a scope, you must stop the scope.

At the target computer command line, you can remove multiple signals from the scope:

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

`signal_index` is optional. If it is left out, all signals are removed.

Examples

The following examples use model `xpcosc`.

Remove all signals from the scope represented by the scope object sc1:

```
tg = slrt;
sc1 = getscope(tg,1);
remsignal(sc1)
```

Remove signals `Integrator1` and `Signal Generator` from the scope on the target computer:

```
tg = slrt;
sc1 = getscope(tg,1);
s0 = getsignalid(tg,'Signal Generator');
s1 = getsignalid(tg,'Integrator1');
remsignal(sc1,[s0,s1])
```

More About

- "Host Scope Usage"

See Also

"Target Computer Commands" | `SimulinkRealTime.hostScope.addsignal` | `SimulinkRealTime.target.getsignalid` | `SimulinkRealTime.target.remscope`

Introduced in R2014a

SimulinkRealTime.hostScope.start

Start execution of host scope on target computer

Syntax

```
start(scope_object)
start(scope_object_vector)
start([scope_object1, scope_object2, . . ., scope_objectN])
```

Arguments

`scope_object`	Name of single vector object.
`scope_object_vector`	Name of vector of scope objects.

Description

`start(scope_object)` starts a scope on the target computer represented by a scope object on the development computer. This method might not start data acquisition, which depends on the trigger settings.

Before using this method, you must create a scope. To create a scope, use the target object method `addscope` or add Simulink Real-Time scope blocks to your Simulink model.

Alternative syntaxes are `start(scope_object_vector)`and `start([scope_object1, scope_object2, . . ., scope_objectN])`.

At the target computer command line, you can use the corresponding command:

```
startscope scope_index
startscope all
```

Examples

Start one scope with the scope object `sc1`:

```
tg = slrt;
sc1 = getscope(tg,1)
start(sc1)
```

Start two scopes, 1 and 2:

```
tg = slrt;
somescopes = getscope(tg,[1,2])
start(somescopes)
```

or

```
tg = slrt;
sc1 = getscope(tg,1)
sc2 = getscope(tg,2)
start([sc1,sc2])
```

Start all scopes:

```
tg = slrt;
allscopes = getscope(tg)
start(allscopes)
```

More About

- "Host Scope Usage"

See Also

"Target Computer Commands" | `SimulinkRealTime.hostScope.stop` | `SimulinkRealTime.target.getscope` | `SimulinkRealTime.target.start`

Introduced in R2014a

SimulinkRealTime.hostScope.stop

Stop execution of host scope on target computer

Syntax

```
stop(scope_object)
stop(scope_object_vector)
stop([scope_object1, scope_object2, . . ., scope_objectN])
```

Arguments

`scope_object`	Name of single vector object.
`scope_object_vector`	Name of vector of scope objects.

Description

`stop(scope_object)` stops a scope on the target computer represented by a scope object on the development computer.

Alternative syntaxes are `stop(scope_object_vector)` and `stop([scope_object1, scope_object2, . . ., scope_objectN])`.

At the target computer command line, you can use the corresponding command:

```
stopscope scope_index
stopscope all
```

Examples

Stop one scope with the scope object `sc1`:

```
tg = slrt;
sc1 = getscope(tg,1)
stop(sc1)
```

Stop two scopes, 1 and 2:

```
tg = slrt;
somescopes = getscope(tg,[1,2])
stop(somescopes)
```

or

```
tg = slrt;
sc1 = getscope(tg,1)
sc2 = getscope(tg,2)
stop([sc1,sc2])
```

Stop all scopes:

```
tg = slrt;
allscopes = getscope(tg)
stop(allscopes)
```

More About

- "Host Scope Usage"

See Also

"Target Computer Commands" | SimulinkRealTime.hostScope.start | SimulinkRealTime.target.getscope | SimulinkRealTime.target.start

Introduced in R2014a

SimulinkRealTime.hostScope.trigger

Software-trigger start of data acquisition for host scope

Syntax

```
trigger(scope_object_vector)
```

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

`trigger(scope_object_vector)` triggers the scope represented by the scope object to acquire the number of data points in the scope object property `NumSamples`.

If the scope object property `TriggerMode` has a value of `'Software'`, this function is the only way to trigger the scope. However, this function can be used on any scope, regardless of trigger mode setting. For example, if a scope is signal triggered and did not trigger because the triggering criteria were not met, this function can be used to force the scope to trigger.

Examples

Using model `xpcosc`, set a single host scope to software trigger, trigger the acquisition of one set of samples, and plot the data on the host from the scope object properties `scope_object.Time` and `scope_object.Data`.

```
tg = slrt;
```

```
tg.StopTime = Inf;
sc1 = addscope(tg,'host',1);
addsignal(sc1, 4)
sc1.TriggerMode = 'software';
start(tg)
start(sc1)
trigger(sc1)
pause(0.5)
plot(sc1.Time, sc1.Data)
stop(sc1)
stop(tg)
```

More About

- "Host Scope Usage"

Introduced in R2014a

SimulinkRealTime.targetScope

Control and access properties of target scopes

Description

The kernel acquires a data package and the scope displays the data on the target computer screen. Depending on the setting of `DisplayMode`, the data is displayed numerically or graphically by a redrawing, sliding, and rolling display.

Methods

The methods in the following table apply to file, host, and target scopes.

Method	Description
`SimulinkRealTime.target`	Add signals to scope represented by scope object
`SimulinkRealTime.target`	Remove signals from scope represented by scope object
`SimulinkRealTime.target`	Start execution of scope on target computer
`SimulinkRealTime.target`	Stop execution of scope on target computer
`SimulinkRealTime.target`	Software trigger start of data acquisition for scope or scopes

Properties

Scope object properties let you select signals to acquire, set triggering modes, and access signal information from the real-time application.

To get the value of a readable scope object property from a scope object:

```
scope_object = getscope(target_object, scope_number);
value = scope_object.scope_object_property
```

For example, to get the `Decimation` of scope `3`:

```
scope_object = getscope(tg, 3);
value = scope_object.Decimation
```

To set the value of a writable scope property from a scope object:

```
scope_object = getscope(target_object, scope_number);
scope_object.scope_object_property = new_value
```

For example, to set the `Decimation` of scope `3`:

```
scope_object = getscope(tg, 3);
scope_object.Decimation = 10
```

Not all properties are user-writable. For example, property `Type` is not writable after you have created the scope.

The properties in the following table apply to file, host, and target scopes.

Property	Description	Writable
`Application`	Name of the Simulink model associated with this scope object.	No
`Decimation`	A number `n`; every `n`th sample is acquired by a scope.	Yes
`NumPrePostSamples`	Number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set `TriggerMode` to `'FreeRun'`, this property has no effect on data acquisition.	Yes
`NumSamples`	Number of contiguous samples captured during the acquisition of a data package. If the scope stops before capturing this number of samples, the scope has the collected data up to the end of data collection, then has zeroes for the remaining uncollected data. Know what type of data you are collecting, because it is possible that your data contains zeroes. For file scopes, this parameter works in conjunction with the **AutoRestart** check box. If you select the **AutoRestart** box, the file scope collects data up to **Number of Samples**, then starts over again, overwriting the buffer. If you do not select the **AutoRestart** box, the file scope collects data only up to **Number of Samples**, then stops.	Yes
`ScopeId`	A numeric index, unique for each scope.	No

Property	Description	Writable
`Signals`	List of signal indices from the target object to display on the scope.	Yes
`Status`	Indicates whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are `'Acquiring'`, `'Ready for being Triggered'`, `'Interrupted'`, and `'Finished'`.	No
`TriggerLevel`	If `TriggerMode` is `'Signal'`, indicates the value the signal has to cross to trigger the scope to start acquiring data. The trigger level can be crossed with a rising or a falling signal.	Yes
`TriggerMode`	Trigger mode for a scope. Valid values are: • `'freerun'` — scope triggers on every sample time. • `'software'` — scope triggers from Command Window. • `'signal'` — scope triggers when a designated signal changes state. • `'scope'` — scope triggers when a designated scope triggers. The default value is `'FreeRun'`.	Yes

Property	Description	Writable
TriggerSample	If TriggerMode is 'Scope', then TriggerSample specifies which sample of the triggering scope to trigger on. For example, if TriggerSample is 0 (default), the current scope triggers on sample 0 (first sample acquired) of the triggering scope. 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. Setting TriggerSample to -1 means that the current scope is triggered at the end of the acquisition cycle of the triggering scope. The first sample of the triggering scope is acquired one sample after the last sample of the triggering scope.	Yes
TriggerScope	If TriggerMode is 'Scope', it identifies the scope to use for a trigger. You can set a scope to trigger when another scope is triggered. Set the slave scope property TriggerScope to the scope index of the master scope.	Yes
TriggerSignal	If TriggerMode is 'Signal', it identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property Signal.	Yes
TriggerSlope	If TriggerMode is 'Signal', it indicates whether the trigger is on a rising signal, falling signal, or either. • either — the signal triggers the scope when it crosses triggerlevel in either the rising or falling directions. • rising — the signal triggers the scope when it crosses triggerlevel in the rising direction. • falling — the signal triggers the scope when it crosses triggerlevel in the falling direction. The default value is 'Either'.	Yes

Property	Description	Writable
`Type`	Determines how the development computer collects and displays its data. - `'Host'` — the data is collected on the target computer and displayed on the development computer. - `'Target'` — the data is collected on the target computer and displayed on the target computer monitor. - `'File'` — the data is collected and stored on the target computer. You set property `Type` only once, when you create the scope on the target computer.	No

The properties in the following table apply only to target scopes.

Property	Description	Writeable
`DisplayMode`	For target scopes, indicate how a scope displays the signals. Values are: - `numerical` — scope displays signal values as text. - `redraw` — scope plots signal values when `numsamples` samples has been acquired. - `rolling` — scope `scope_index` plots signal values at every sample time. **Note:** Value `sliding` will be removed in a future release. It behaves like value `rolling`. For host or file scopes, this parameter has no effect.	Yes

Property	Description	Writeable
	.	
`Grid`	Values are `'on'` and `'off'`. For host or file scopes, this parameter has no effect.	Yes
`Mode`	**Note:** The `Mode` property will be removed in a future release. • For target scopes, use `DisplayMode`. • For file scopes, use `WriteMode`. • For host scopes, this parameter has no effect.	Yes
`YLimit`	Minimum and maximum *y*-axis values. This property can be set to `'auto'`. For host or file scopes, this parameter has no effect.	Yes

More About

- "Target Scope Usage"

Introduced in R2014a

SimulinkRealTime.targetScope.addsignal

Add signals to target scope represented by scope object

Syntax

```
addsignal(scope_object_vector, signal_index_vector)
```

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.

Description

`addsignal(scope_object_vector, signal_index_vector)` adds signals to a scope object. The signals must be specified by their indices, which you can retrieve using the target object method `SimulinkRealTime.target.getsignalid`. If `scope_object_vector` has two or more scope objects, the same signals are assigned to each scope.

Before you can add a signal to a scope, you must stop the scope.

At the target computer command line, you can add multiple signals to the scope:

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

Examples

The following examples use model `xpcosc`.

Add signals `Integrator1` and `Signal Generator` to scope object `sc1`.

```
tg = slrt;
sc1 = addscope(tg,'target',1);
s0 = getsignalid(tg,'Signal Generator');
s1 = getsignalid(tg,'Integrator1');
addsignal(sc1,[s0,s1]);
```

The scope object property `Signals` is updated to include the added signals. Type `sc1` to display the properties and values for scope `sc1`.

More About

- "Target Scope Usage"

See Also

"Target Computer Commands" | `SimulinkRealTime.target.addscope` | `SimulinkRealTime.target.getsignalid` | `SimulinkRealTime.targetScope.remsignal`

Introduced in R2014a

SimulinkRealTime.targetScope.remsignal

Remove signals from target scope represented by scope object

Syntax

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

Arguments

`scope_object_vector` Scope object or vector of scope objects. The target object methods `addscope` or `getscope` create scope objects.

`signal_index_vector` Index numbers from the scope object property `Signals`. This argument is optional. If it is left out, all signals are removed.

Description

`remsignal(scope_object)` removes all signals from a scope object.

`remsignal(scope_object, signal_index_vector)` 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 `scope_object` is a vector containing two or more scope objects, the same signals are removed from each scope.

Before you can remove a signal from a scope, you must stop the scope.

At the target computer command line, you can remove multiple signals from the scope:

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

`signal_index` is optional. If it is left out, all signals are removed.

Examples

The following examples use model `xpcosc`.

Remove all signals from the scope represented by the scope object sc1.

```
tg = slrt;
sc1 = getscope(tg,1);
remsignal(sc1)
```

Remove signals `Integrator1` and `Signal Generator` from the scope on the target computer.

```
tg = slrt;
sc1 = getscope(tg,1);
s0 = getsignalid(tg,'Signal Generator');
s1 = getsignalid(tg,'Integrator1');
remsignal(sc1,[s0,s1])
```

More About

- "Target Scope Usage"

See Also

"Target Computer Commands" | `SimulinkRealTime.target.getsignalid` | `SimulinkRealTime.target.remscope` | `SimulinkRealTime.targetScope.addsignal`

Introduced in R2014a

SimulinkRealTime.targetScope.start

Start execution of target scope on target computer

Syntax

```
start(scope_object)
start(scope_object_vector)
start([scope_object1, scope_object2, . . ., scope_objectN])
```

Arguments

`scope_object`	Name of single vector object.
`scope_object_vector`	Name of vector of scope objects.

Description

`start(scope_object)` starts a scope on the target computer represented by a scope object on the development computer. This method might not start data acquisition, which depends on the trigger settings.

Before using this method, you must create a scope. To create a scope, use the target object method `addscope` or add Simulink Real-Time scope blocks to your Simulink model.

Alternative syntaxes are `start(scope_object_vector)`and `start([scope_object1, scope_object2, . . ., scope_object`*N*`])`.

At the target computer command line, you can use the corresponding command:

```
startscope scope_index
startscope all
```

Examples

Start one scope with the scope object `sc1`.

```
tg = slrt;
sc1 = getscope(tg,1)
start(sc1)
```

Start two scopes, 1 and 2.

```
tg = slrt;
somescopes = getscope(tg,[1,2])
start(somescopes)
```

or

```
tg = slrt;
sc1 = getscope(tg,1)
sc2 = getscope(tg,2)
start([sc1,sc2])
```

Start all scopes:

```
tg = slrt;
allscopes = getscope(tg)
start(allscopes)
```

More About

- “Target Scope Usage”

See Also

“Target Computer Commands” | `SimulinkRealTime.target.getscope` | `SimulinkRealTime.target.start` | `SimulinkRealTime.targetScope.stop`

Introduced in R2014a

SimulinkRealTime.targetScope.stop

Stop execution of target scope on target computer

Syntax

```
stop(scope_object)
stop(scope_object_vector)
stop([scope_object1, scope_object2, . . ., scope_objectN])
```

Arguments

`scope_object`	Name of single vector object.
`scope_object_vector`	Name of vector of scope objects.

Description

`stop(scope_object)` stops a scope on the target computer represented by a scope object on the development computer.

Alternative syntaxes are `stop(scope_object_vector)` and `stop([scope_object1, scope_object2, . . ., scope_objectN])`.

At the target computer command line, you can use the corresponding command:

```
stopscope scope_index
stopscope all
```

Examples

Stop one scope with the scope object `sc1`.

```
tg = slrt;
sc1 = getscope(tg,1)
stop(sc1)
```

Stop two scopes, 1 and 2.

```
tg = slrt;
somescopes = getscope(tg,[1,2])
stop(somescopes)
```

or

```
tg = slrt;
sc1 = getscope(tg,1)
sc2 = getscope(tg,2)
stop([sc1,sc2])
```

Stop all scopes:

```
tg = slrt;
allscopes = getscope(tg)
stop(allscopes)
```

More About

- “Target Scope Usage”

See Also

“Target Computer Commands” | `SimulinkRealTime.target.getscope` | `SimulinkRealTime.target.start` | `SimulinkRealTime.targetScope.start`

Introduced in R2014a

SimulinkRealTime.targetScope.trigger

Software-trigger start of data acquisition for target scope

Syntax

```
trigger(scope_object_vector)
```

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

`trigger(scope_object_vector)` triggers the scope represented by the scope object to acquire the number of data points in the scope object property `NumSamples`.

If the scope object property `TriggerMode` has a value of `'Software'`, this function is the only way to trigger the scope. However, this function can be used on any scope, regardless of trigger mode setting. For example, if a scope is signal triggered and did not trigger because the triggering criteria were not met, this function can be used to force the scope to trigger.

Examples

Using model `xpcosc`, set a single target scope to software trigger, trigger the acquisition of one set of samples, and display the data on the target screen.

```
tg = slrt;
tg.StopTime = Inf;
sc1 = addscope(tg,'target',1);
```

```
addsignal(sc1, 4)
sc1.TriggerMode = 'software';
start(tg)
start(sc1)
trigger(sc1)
pause(0.5)
stop(sc1)
stop(tg)
```

More About

- “Target Scope Usage”

Introduced in R2014a