Simulink® Real-Time™
API Guide

R2014**a**

MATLAB®&SIMULINK®

MathWorks®

How to Contact MathWorks

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup
`www.mathworks.com/contact_TS.html`	Technical Support
`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

Simulink® Real-Time™ API Guide

© COPYRIGHT 2002–2014 by The MathWorks, Inc.

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

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

Trademarks

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

Patents

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

Revision History

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)

Contents

1 Introduction

Simulink Real-Time APIs 1-2

Simulink Real-Time API for Microsoft .NET Framework 1-3

Simulink Real-Time C API 1-5

Required Products 1-7

2 Simulink Real-Time API for Microsoft .NET Framework

Using the Simulink Real-Time API for .NET Framework 2-2
- Features and Benefits 2-2
- xpcosc Client Applications 2-3
- File Server Browser Client Application 2-3

Simulink Real-Time .NET API Object Model 2-4

Simulink Real-Time API for .NET Framework Classes 2-5
- Mathworks.xPCTarget.Framework.xPCTargetPC 2-5
- Mathworks.xPCTarget.Framework.xPCApplication 2-6
- Mathworks.xPCTarget.Framework.xPCScopes 2-6
- Mathworks.xPCTarget.Framework.xPCParameters 2-6
- Mathworks.xPCTarget.Framework.xPCParameter 2-6
- Mathworks.xPCTarget.Framework.xPCSignals 2-7
- Mathworks.xPCTarget.Framework.xPCSignal 2-7

Mathworks.xPCTarget.Framework.xPCAppLogger **2-7**

Simulink Real-Time .NET API Usage **2-8**

Simulink Real-Time .NET API Application Deployment **2-10**

3 Simulink Real-Time API for C

Using the C API **3-2**

Visual C Console Application **3-4**
Target Application **3-4**
Folders and Files **3-4**
Building the Simulink Real-Time Application **3-6**
Creating a Visual C Application **3-6**
Building a Visual C Application **3-9**
Running an Simulink Real-Time Visual C API Application **3-10**
Using the Simulink Real-Time C API Application **3-10**
C Code for sf_car_xpc.c **3-17**

4 Simulink Real-Time .NET API Examples

Visual Basic GUI Using .NET **4-2**
Before Starting **4-2**
Accessing the Demo Project Solution **4-3**
Rebuilding the Demo Project Solution **4-3**
Using the Demo Executable **4-4**

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

Simulink Real-Time API for Microsoft .NET Framework — Alphabetical List . . . 5-2

6 Simulink Real-Time API Reference for C

C API Error Messages . . . 6-2

C API Structures and Functions — Alphabetical List . . 6-6

7 Simulink Real-Time API Reference for COM

COM API Methods — Alphabetical List . . . 7-2

8 MATLAB API

MATLAB API — Alphabetical List . . . 8-2

Introduction

- "Simulink® Real-Time™ APIs" on page 1-2
- "Simulink® Real-Time™ API for Microsoft .NET Framework" on page 1-3
- "Simulink® Real-Time™ C API" on page 1-5
- "Required Products" on page 1-7

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 target 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 host computer and the target computer via an Ethernet or serial connection
- Load the target application, a `.dlm` file, to the target computer
- Run that application on the target computer
- Monitor the behavior of the target application on the target computer
- Stop that application on the target computer
- Unload the target 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 1-5

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 methods and properties that allow you to manipulate and interact with it. The API provides a number of classes, including those for target applications, scopes, the file system, and the target computer. The xPCTargetPC class is the main class that sits on top of a hierarchy of classes. This document presents the API reference. 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.

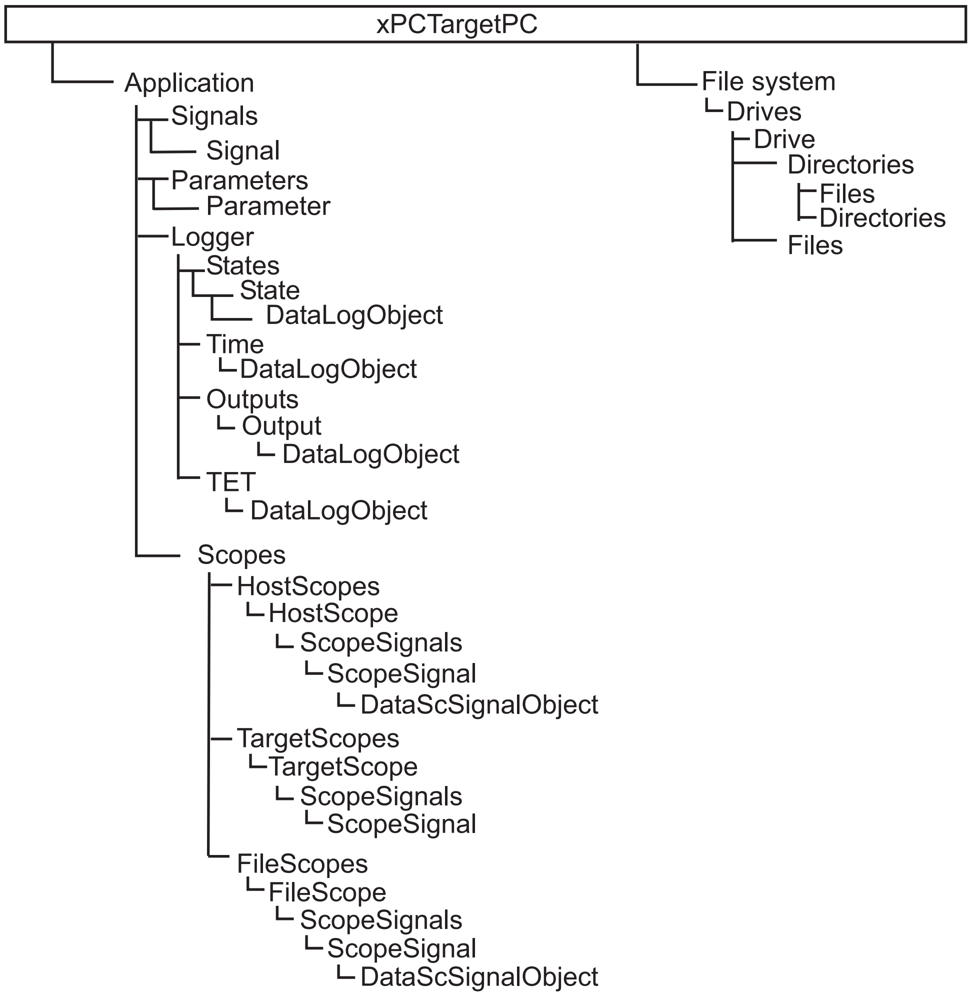
The Simulink Real-Time .NET object model closely models the Simulink Real-Time system. One `xPCTargetPC Class` object represents one Simulink Real-Time system.

An `xPCApplication Class` object represents the target application. It contains xPCSignals, xPCParameters, and xPC*Logger objects. These objects respectively represent the signals, parameters, and logs available in the target application.

An `xPCFileSystem Class` object represents the entire Simulink Real-Time file system. It contains objects like the following:

- xPCDriveInfo, which represents a volume drive that the target computer recognizes.
- xPCDirectoryInfo, which represents a target computer folder item.
- xPCFileInfo, which represents a target computer file item.

The following graphic outlines the xPCTargetPC hierarchy.


xPCTargetPC
Application
Signals
Signal
Parameters
Parameter
Logger
States
State
DataLogObject
Time
DataLogObject
Outputs
Output
DataLogObject
TET
DataLogObject
Scopes
HostScopes
HostScope
ScopeSignals
ScopeSignal
DataScSignalObject
TargetScopes
TargetScope
ScopeSignals
ScopeSignal
FileScopes
FileScope
ScopeSignals
ScopeSignal
DataScSignalObject
File system
Drives
Drive
Directories
Files
Directories
Files

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 an 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 an 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 target 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 host and target computer, a default timeout of 5 seconds controls how long a target computer can take to communicate with a host 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.

Required Products

Refer to System Requirements for a list of the required Simulink Real-Time products. In addition, you need 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.

Simulink Real-Time API for Microsoft .NET Framework

- “Using the Simulink® Real-Time™ API for .NET Framework” on page 2-2
- “Simulink® Real-Time™ .NET API Object Model” on page 2-4
- “Simulink® Real-Time™ API for .NET Framework Classes” on page 2-5
- “Simulink® Real-Time™ .NET API Usage” on page 2-8
- “Simulink® Real-Time™ .NET API Application Deployment” on page 2-10

Using the Simulink Real-Time API for .NET Framework

The Simulink Real-Time API for .NET framework is a fully managed .NET framework component. Although this framework is designed to work with the Microsoft Visual Studio software, you can use it with other development environments that support the .NET framework. This API is a fully programmable tool set. It contains easy-to-use components and types that enable you to quickly design Simulink Real-Time client applications. You can use this API with a programming language that supports .NET technology.

In this section...

"Features and Benefits" on page 2-2

"xpcosc Client Applications" on page 2-3

"File Server Browser Client Application" on page 2-3

Features and Benefits

The Simulink Real-Time API for .NET framework includes the following features and benefits:

- 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)

xpcosc Client Applications

The `Simple Client Application with the .NET API` example illustrates how to use the Simulink Real-Time API for Microsoft .NET Framework to create client applications to interface with the `xpcosc` model downloaded on the target computer. This example provides two client applications:

- `Example1` — Illustrates a client application that runs on the host computer. The client application provides a GUI through which you can enter the IP address port of the target computer with which you want to connect. It consists of the toolbox items:
  - Buttons
  - TextBoxes
  - TrackBar
- `Example2` — In addition to the same toolbox controls as Example 1, this example also contains a chart that displays signals from the `xpcosc` target application.

File Server Browser Client Application

The API Simulink Real-Time API for the .NET Framework has the following example, located in:

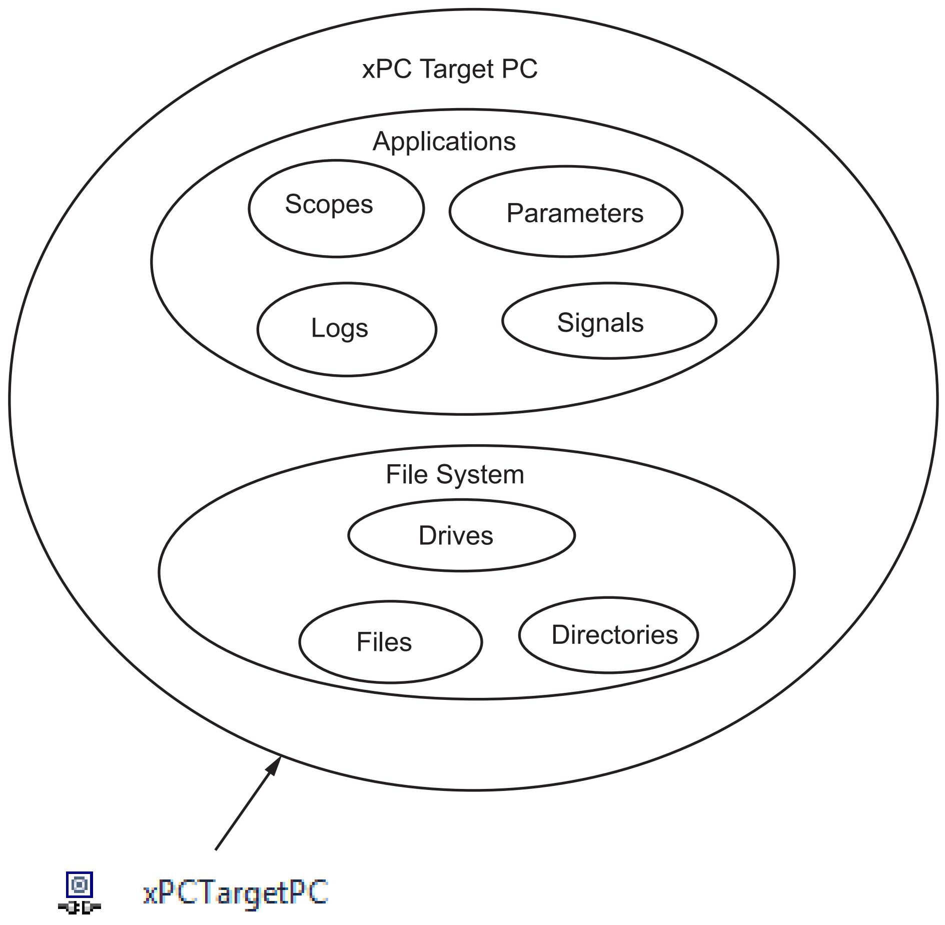
```
matlabroot\toolbox\rtw\targets\xpc\api\xPCFrameworkSamples\FileSystemBrowser
```

This example illustrates how to use the Simulink Real-Time API for the .NET Framework to create a file browser to browse folders and files on the target computer file system. The application resides on the host computer and connects to the target computer to browse its file system.

This is a C# application project developed with the Microsoft Visual Studio 2008 IDE. It illustrates how to build a standalone Simulink Real-Time executable to connect to a target computer and a host computer. See the `Readme.txt` file in the example folder for instructions on how to access and build the example code.

Simulink Real-Time .NET API Object Model

To develop solutions that use the Simulink Real-Time .NET API, you can interact with the API objects in the Simulink Real-Time .NET API object model. The object model corresponds to structure of the Simulink Real-Time environment. The object model is hierarchical and straightforward. The following is a conceptual view of the xPCTargetPC object.



Simulink Real-Time API for .NET Framework Classes

The Simulink Real-Time .NET API provides an expansive object model layer. You should start your client model development on the following objects:

In this section...

"Mathworks.xPCTarget.Framework.xPCTargetPC" on page 2-5

"Mathworks.xPCTarget.Framework.xPCApplication" on page 2-6

"Mathworks.xPCTarget.Framework.xPCScopes" on page 2-6

"Mathworks.xPCTarget.Framework.xPCParameters" on page 2-6

"Mathworks.xPCTarget.Framework.xPCParameter" on page 2-6

"Mathworks.xPCTarget.Framework.xPCSignals" on page 2-7

"Mathworks.xPCTarget.Framework.xPCSignal" on page 2-7

"Mathworks.xPCTarget.Framework.xPCAppLogger" on page 2-7

Mathworks.xPCTarget.Framework.xPCTargetPC

The xPCTargetPC object represents the overall Simulink Real-Time environment system. It is at the root level of the object model and exposes information about the Simulink Real-Time session after connecting to your target computer. It provides many class member functions that you use to access information and manipulate its behavior.

The xPCTargetPC object principally supports a run-time user-driven mode of execution. However, the xPCTargetPC type is also a .NET component implementation that supports an optional developer-driven model of execution, a design-time capability. You can integrate the design-time capability with the Microsoft Visual Studio IDE. It supports creation and management of the xPCTargetPC component. With this capability, you can perform the following operations with xPCTargetPC components

- Drag and drop into the form design
- Property configuration
- Delete from the form design

Design-time support includes a properties window in which you can configure design-time members, code serialization, and property-editing support with UI type editors. This supports enables you to build Simulink Real-Time application quickly and effortlessly by dragging the component and using its functionality as required. For more information on using Microsoft Visual Studio .NET, see http://msdn.microsoft.com/en-us/library/aa973739(v=vs.71).aspx.

Mathworks.xPCTarget.Framework.xPCApplication

The xPCApplication object represents the Simulink Real-Time real-time application that you generate from a Simulink model and download to the target computer. The xPCApplication object exposes information and properties of the target application. It also contains members you need to:

- Access application information
- Manipulate application behavior
- Return other objects such as child components of the application

Mathworks.xPCTarget.Framework.xPCScopes

The xPCScopes object represents a container or place holder to access and interface with Simulink Real-Time scopes. This object enables advanced signal data acquisition techniques. With this object, you can access child objects related to scopes.

Mathworks.xPCTarget.Framework.xPCParameters

The xPCParameters object represents a container or place holder to access application parameters. You can access xPCParameter objects with this object.

Mathworks.xPCTarget.Framework.xPCParameter

The xPCParameter object represents a specific application parameter, which represents a run-time parameter of a specific block. With this object, you can access information related to the block parameter. With this object, you can also tune parameter values during simulation.

Mathworks.xPCTarget.Framework.xPCSignals

The xPCSignals object represents a container or place holder to access the application signals. With this object, you can access xPCSignal objects.

Mathworks.xPCTarget.Framework.xPCSignal

The xPCSignal object represents a specific application signal, which represents the port signal of a non-graphical block output. With this object, you can access information related to the signal. It also allows you to monitor signal behavior during simulation.

Mathworks.xPCTarget.Framework.xPCAppLogger

The xPCAppLogger object represents a place holder for logging objects. It contains members that return specific logging objects.

Simulink Real-Time .NET API Usage

This topic presents the Simulink Real-Time API for .NET framework reference using the C# language and the Microsoft Visual Studio environment. At a minimum:

- Use the xPCTargetPC component in the Visual Studio environment. This addition provides convenient design-time features. To do this:

  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.

     The xPCTargetPC control makes available in the Visual Studio **Properties** window its data and appearance properties. You can click the xPCTargetPC control in the design surface to explore and customize the xPCTargetPC properties.

- Add a reference for `xPCFramework.dll` to your project (for example, to create a console application), include the following in your code. Doing so enables you to access the types available from the Simulink Real-Time environment

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

- 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. You also need this file to execute the application.

  The Simulink Real-Time library has a 32-bit and a 64-bit version of the `xpcapi.dll`.

Note On 64-bit platforms, if you build a 64–bit target application in the Microsoft Visual Studio environment, and want to use the xPCTargetPC nonvisual component; place the 32-bit version of `xpcapi.dll` in the solution folder and 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.

- Do not test communication between host and target computers (`xPCTargetPC.Ping` method) until you have connected to the target computer (`xPCTargetPC.Connect` method).

Note Be sure to disconnect the target computer from the host computer before starting .NET client applications. A target computer can be connected to only one host computer at a time. You can use `slrtpingtarget` to verify connectivity; this function disconnects from the target computer when done.

Simulink Real-Time .NET API Application Deployment

This topic describes guidelines when distributing your Simulink Real-Time API for Microsoft .NET Framework GUI application:

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

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

Simulink Real-Time API for C

- “Using the C API” on page 3-2
- “Visual C Console Application” on page 3-4

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
- If you want to rebuild the model (`sf_car_xpc`), or otherwise use the MATLAB environment, you must have Simulink Real-Time Version 2.0 or later. To determine the version of Simulink Real-Time you are currently using, at the MATLAB command line, type

  ```
  slrtlib
  ```

  This opens the Simulink Real-Time Simulink blocks library. The version of Simulink Real-Time should be at the bottom of the window.
- You can work with Simulink Real-Time applications with either MATLAB or an Simulink Real-Time C API application. If you are working with an 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 “C API Structures and Functions — Alphabetical List” 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`.

Visual C Console Application

This topic shows how to use the Simulink Real-Time C API to create a Win32 console application written in C. You can use this example as a template to write your own application.

In this section...

"Target Application" on page 3-4

"Folders and Files" on page 3-4

"Building the Simulink® Real-Time™ Application" on page 3-6

"Creating a Visual C Application" on page 3-6

"Building a Visual C Application" on page 3-9

"Running an Simulink® Real-Time™ Visual C API Application" on page 3-10

"Using the Simulink® Real-Time™ C API Application" on page 3-10

"C Code for sf_car_xpc.c" on page 3-17

Target Application

Before you start, you should have an existing Simulink Real-Time application that you want to load and run on a target computer. The following topics use the target application `sf_car_xpc.dlm`, built from the Simulink model `sf_car_xpc`, which models an automatic transmission control system. The automatic transmission control system consists of modules that represent the engine, transmission, and vehicle, with an additional logic block to control the transmission ratio. User inputs to the model are in the form of throttle (%) and brake torque (pound-foot). You can control the target application through MATLAB with the Simulink External Mode interface, or through a custom Simulink Real-Time C API application.

Folders and Files

This folder contains the C source of a Win32 console application that serves as an example for using the Simulink Real-Time C API. The `sf_car_xpc` files are in the folder:

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

Filename	Description
`VisualBasic\Models\-sf_car_xpc\sf_car_xpc`	Simulink model for use with Simulink Real-Time
`VisualBasic\Models\-sf_car_xpc\sf_car_xpc.dlm`	Target application compiled from Simulink model
`VisualC\sf_car_xpc.dsp`	Project file for API application
`sf_car_xpc.c`	Source code for API application
`VisualC\sf_car_xpc.exe`	Compiled API application
`VisualBasic\Models\-xpcapi.dll`	Simulink Real-Time C API functions for supported programming languages. Place this file in one of the following, in order of preference: • Folder from which the application is loaded • Windows system folder

The Simulink Real-Time C API files are in the folder:

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

You will need the files listed below for creating your own API application with Microsoft Visual C++®.

Filename	Description
`xpcapi.h`	Mapping of data types between Simulink Real-Time C API and Visual C
`xpcapiconst.h`	Symbolic constants for using scope, communication, and data-logging functions
`xpcinitfree.c`	C functions to upload API from `xpcapi.dll`
`xpcapi.dll`	Simulink Real-Time C API functions for supported programming languages

Building the Simulink Real-Time Application

These tutorials use the prebuilt Simulink Real-Time application:

```
matlabroot\toolbox\rtw\targets\
xpc\api\VisualC\sf_car_xpc.dlm
```

You can rebuild this application for your example:

1. Create a new folder under your MathWorks® folder. For example,

   ```
   D:\mwd\sf_car_xpc2
   ```

2. Create a Simulink model and save to this folder. For example,

   ```
   sf_car_xpc2
   ```

3. Build the target application with Simulink Coder™ and Microsoft Visual C++. The target application file `sf_car_xpc2.dlm` is created.

Using Another C/C++ Compiler

These tutorials describe how to create and build C applications using Microsoft Visual C++. However, to build an Simulink Real-Time C API application, you can use other C/C++ compilers, provided they are capable of generating a Win32 application. You will need to link and compile the Simulink Real-Time C API application along with `xpcinitfree.c` to generate the executable. The file `xpcinitfree.c` contains the definitions for the files in the Simulink Real-Time C API and is located:

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

Creating a Visual C Application

This tutorial describes how to create a Visual C application. It is assumed that you know how to write C applications. Of particular note when writing Simulink Real-Time C API applications,

- Call the function `xPCInitAPI` at the start of the application to load the functions.
- Call the function `xPCFreeAPI` at the end of the application to free the memory allocated to the functions.

To create a C application with a program such as Microsoft Visual C++,

1 From the previous tutorial, change folder to the new folder. This is your working folder. For example,

   ```
   D:\mwd\sf_car_xpc2
   ```

2 Copy the files `xpcapi.h`, `xpcapi.dll`, `xpcapiconst.h`, and `xpcintfree.c` to the working folder. For example,

   ```
   D:\mwd\sf_car_xpc2
   ```

3 Click the **Start** button, choose the **All Programs** option, and choose the **Microsoft Visual C++** entry. Select the **Microsoft Visual C++** option.

   The Microsoft Visual C++ application is displayed.

4 From the **File** menu, click **New**.

5 At the New dialog box, click the **File** tab.

6 In the left pane, select **C++ Source File**. In the right, enter the name of the file. For example, `sf_car_xpc.c`. Select the folder. For example, `C:\mwd\sf_car_xpc2`.

7 Click **OK** to create this file.

8 Enter your code in this file. For example, you can enter the contents of `sf_xpc_car.c` into this file.

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

10 At the New dialog box, click the **Projects** tab.

11 In the left pane, select **Win32 Console Application**. On the right, enter the name of the project. For example, `sf_car_xpc`. Select the working folder from step 1. For example, `C:\mwd\sf_car_xpc2`.

12 To create the project, click **OK**.

   A Win32 Console Application dialog box is displayed.

13 To create an empty project, select **An empty project**.

14 Click **Finish**.

15 To confirm the creation of an empty project, click **OK** at the following dialog box.

16 To add the C file you created in step 7, from the **Project** menu, select the **Add to Project** option and select **Files**.

17 Browse for the C file you created in step 7. For example,

```
D:\mwd\sf_car_xpc2\sf_car_xpc.c
```

Click **OK**.

18 Browse for the `xpcinitfree.c` file. For example, `D:\mwd\xpcinitfree.c`. Click **OK**.

Note The code for linking in the functions in `xpcapi.dll` is in the file `xpcinitfree.c`. You must compile and link `xpcinitfree.c` with your custom application for it to load `xpcapi.dll` at execution time.

---

19 If you did not copy the files `xpcapi.h`, `xpcapi.dll`, and `xpcapiconst.h` into the working or project folder, you should either copy them now, or also add these files to the project.

20 From the **File** menu, click **Save Workspace**.

When you are ready to build your C application, go to "Building a Visual C Application" on page 3-9.

Placing the Target Application File in a Different Folder

The `sf_car_xpc.c` file assumes that the Simulink Real-Time application file `sf_car_xpc.dlm` is in the same folder as `sf_car_xpc.c`. If you move that target application file (`sf_car_xpc.dlm`) to a new location, change the path to this file in the API application (`sf_car_xpc.c`) and recompile the API application. The relevant line in `sf_car_xpc.c` is in the function `main()`, and looks like this:

```
xPCLoadApp(port, ".", "sf_car_xpc"); checkError("LoadApp: ");
```

The second argument (`"."`) in the call to `xPCLoadApp` is the path to `sf_car_xpc.dlm`. The `"."` indicates that the files `sf_car_xpc.dlm` and `sf_car_xpc.c` are in the same folder. If you move the target application, enter its new path and rebuild the Simulink Real-Time C API application.

Building a Visual C Application

This tutorial describes how to build the Visual C application from the previous tutorial, or to rebuild the example executable `sf_car_xpc.exe`, using Microsoft Visual C++:

1. To build your own application using the Simulink Real-Time C API, copy the files `xpcapi.h`, `xpcapi.dll`, `xpcapiconst.h`, and `xpcinitfree.c` into the working or project folder.
2. If Microsoft Visual C++ is not already running, click the **Start** button, choose the **All Programs** option, and choose the **Microsoft Visual C++** option.
3. From the **File** menu, click **Open**.

   The Open dialog box is displayed.
4. Use the browser to select the project file for the application you want to build. For example, `sf_car_xpc.dsp`.
5. If a corresponding workspace file (for example, `sf_car_xpc.dsw`) exists for that project, a dialog box prompts you to open that workspace instead. Click **OK**.
6. Build the application for the project. From the **Build** menu, select either the **Build** `project_name.exe` or **Rebuild All** option.

Microsoft Visual C++ creates a file named `project_name.exe`, where `project_name` is the name of the project.

When you are ready to run your Visual C Application, go to "Running an Simulink® Real-Time™ Visual C API Application" on page 3-10.

Running an Simulink Real-Time Visual C API Application

Before starting the API application `sf_car_xpc.exe`, verify the following:

- The file `xpcapi.dll` must either be in the same folder as the Simulink Real-Time C API application executable, or it must be in the Windows system folder (typically `C:\windows\system` or `C:\winnt\system32`) for global access. The Simulink Real-Time C API application depends on this file, and will not run if the file is not found. The same is true for other applications you write using Simulink Real-Time C API functions.
- The compiled target application `sf_car_xpc.dlm` must be in the same folder as the Simulink Real-Time C API executable. Do not move this file out of this folder. Moving the file requires you to change the path to the target application in the API application and recompile, as described in "Building a Visual C Application" on page 3-9.

Using the Simulink Real-Time C API Application

To run a Simulink Real-Time C API application, you must have a working target computer running at least Simulink Real-Time Version 2.0 (Release 13).

This tutorial assumes that you are using the Simulink Real-Time C API application `sf_car_xpc.exe` that comes with Simulink Real-Time. In turn, `sf_car_xpc.exe` expects that the Simulink Real-Time application is `sf_car_xpc.dlm`.

If you are going to run a version of `sf_car_xpc.exe` that you compiled yourself using the `sf_car_xpc.c` code that comes with Simulink Real-Time, you can run that application instead. Verify the following files are in the same folder:

- `sf_car_xpc.exe`, the Simulink Real-Time C API executable
- `sf_car_xpc.dlm`, the Simulink Real-Time application to be loaded to the target computer
- `xpcapi.dll`, the Simulink Real-Time C API dynamic link library

  If you copy this file to the Windows system folder, you do not need to provide this file in the same folder.

How to Run the sf_car_xpc Executable

1. Create an Simulink Real-Time boot disk with a serial or network communication. If you use serial communications, set the baud rate to `115200`. Otherwise, create the boot disk as directed in Simulink Real-Time Getting Started.

2. Start the target computer with the Simulink Real-Time boot disk.

   The target computer displays messages like the following in the top rightmost message area.

   ```
   System:  Host-Target Interface is RS232 (COM1/2)
   ```

   or

   ```
   System:  Host-Target Interface is TCP/IP (Ethernet)
   ```

3. If you have downloaded target applications to the target computer through MATLAB, in the MATLAB window, type

   ```
   close(slrt)
   ```

   This command disconnects MATLAB from the target computer and leaves the target computer ready to connect to another client.

4. On the host computer, open a DOS window. Change folder to:

   ```
   C:\matlabroot\toolbox\rtw\targets\xpc\api\VisualC
   ```

   If you are running your own version of `sf_car_xpc.exe`, change to the folder that contains the executable and Simulink Real-Time application. For example,

   ```
   D:\mwd\sf_car_xpc2
   ```

5. From that DOS window, enter the command to start the example application on the host computer and download the target application to the target computer.

   The syntax for the example command is

```
sf_car_xpc {-t IpAddress:IpPort|-c COMport}
```

If you set up the Simulink Real-Time boot disk to use TCP/IP, then give the target computer's IP address and IP port as arguments to `sf_car_xpc`, along with the option `-t`. For example, at the DOS prompt, type

```
sf_car_xpc -t 192.168.0.1:22222
```

If you set up the Simulink Real-Time boot disk to use RS-232, give the serial port number as a command-line option. Note that indexing of serial ports starts from 0 instead of 1. For example, if you are using serial communication from COM port 1 on the host computer, type

```
  sf_car_xpc -c 0
```

On the host computer, the example application displays the following message:

```
*-------------------------------------------------------------*
*          Simulink Real-Time API Demo: sf_car_xpc.                  *
*                                                             *
* Copyright (c) 2000 The MathWorks, Inc. All Rights Reserved. *
*-------------------------------------------------------------*
Application sf_car_xpc loaded. SampleTime 0.001 StopTime: -1
R  Br  Th G  VehSpeed   VehRPM
- ---- -- - ---------- ---------
N    0  0 0    0.000    1000.000
```

The relevant line here is the last one, which displays the status of the application. The headings are as follows:

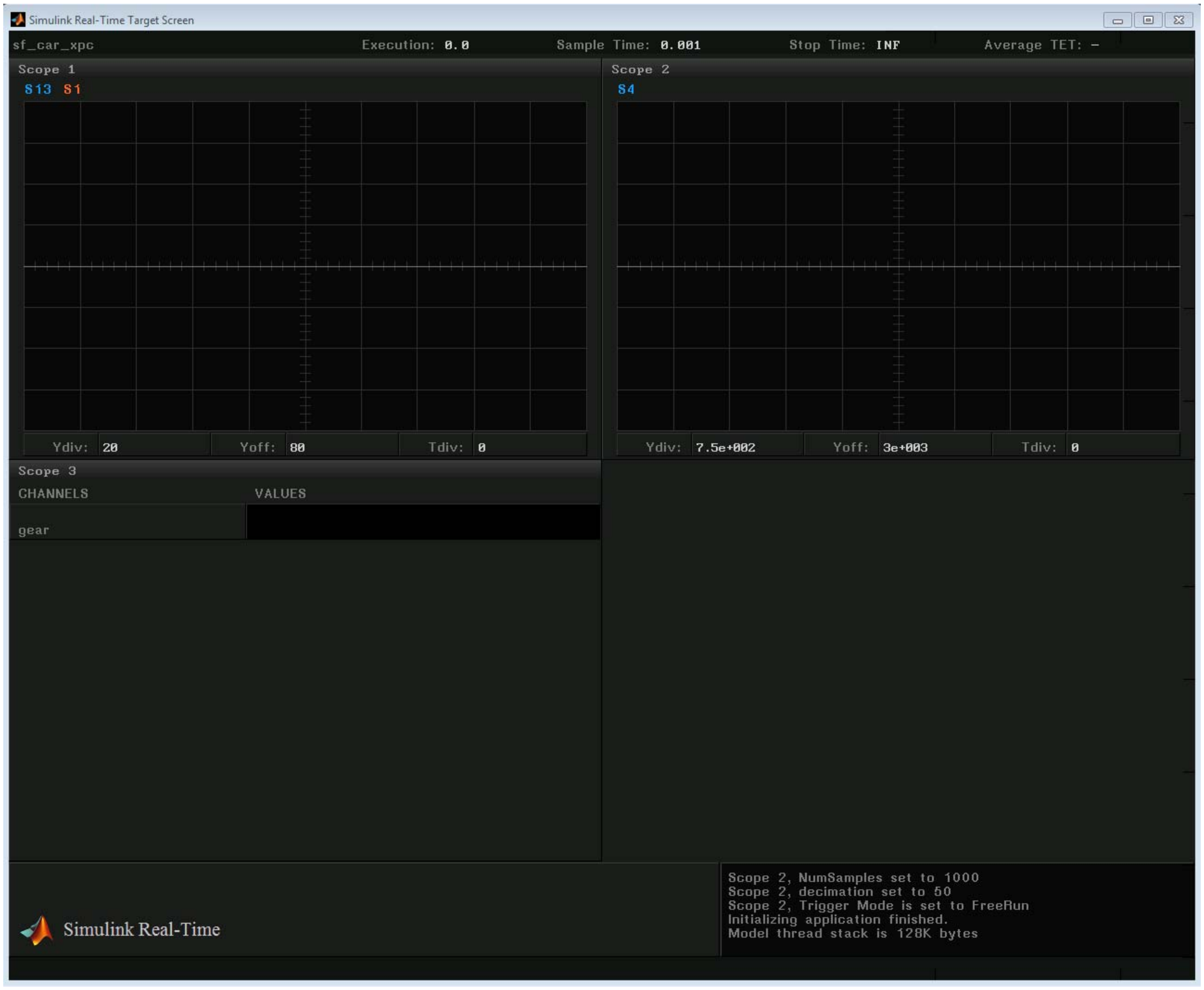
R	The status of the target application: R if running, N if stopped
Br	The brake torque; legal values range from 0 to 4000
Th	The throttle as a percentage (0 - 100) of the total
G	Gear the vehicle is in (ranges between 1 and 4)
VehSpeed	Speed of the vehicle in miles per hour
VehRPM	Revolutions per minute of the vehicle engine (0 to 6000)

From this screen, various keystrokes control the target application. The following list summarizes these keys:

Key	Action
s	Start or stop the application, depending on whether the application is active or not.
T	Increase the throttle by 1 (does not go above 100).
t	Decrease the throttle by 1 (does not go below 0).
B	Increase the brake value by 20 (does not go above 4000).
b	Decrease the brake value by 20 (does not go below 0).
Q or **Ctrl+C**	Quit the application.

Note Note that a positive value for the brake automatically sets the throttle value to 0, and a positive value for the throttle automatically sets the brake value to 0.

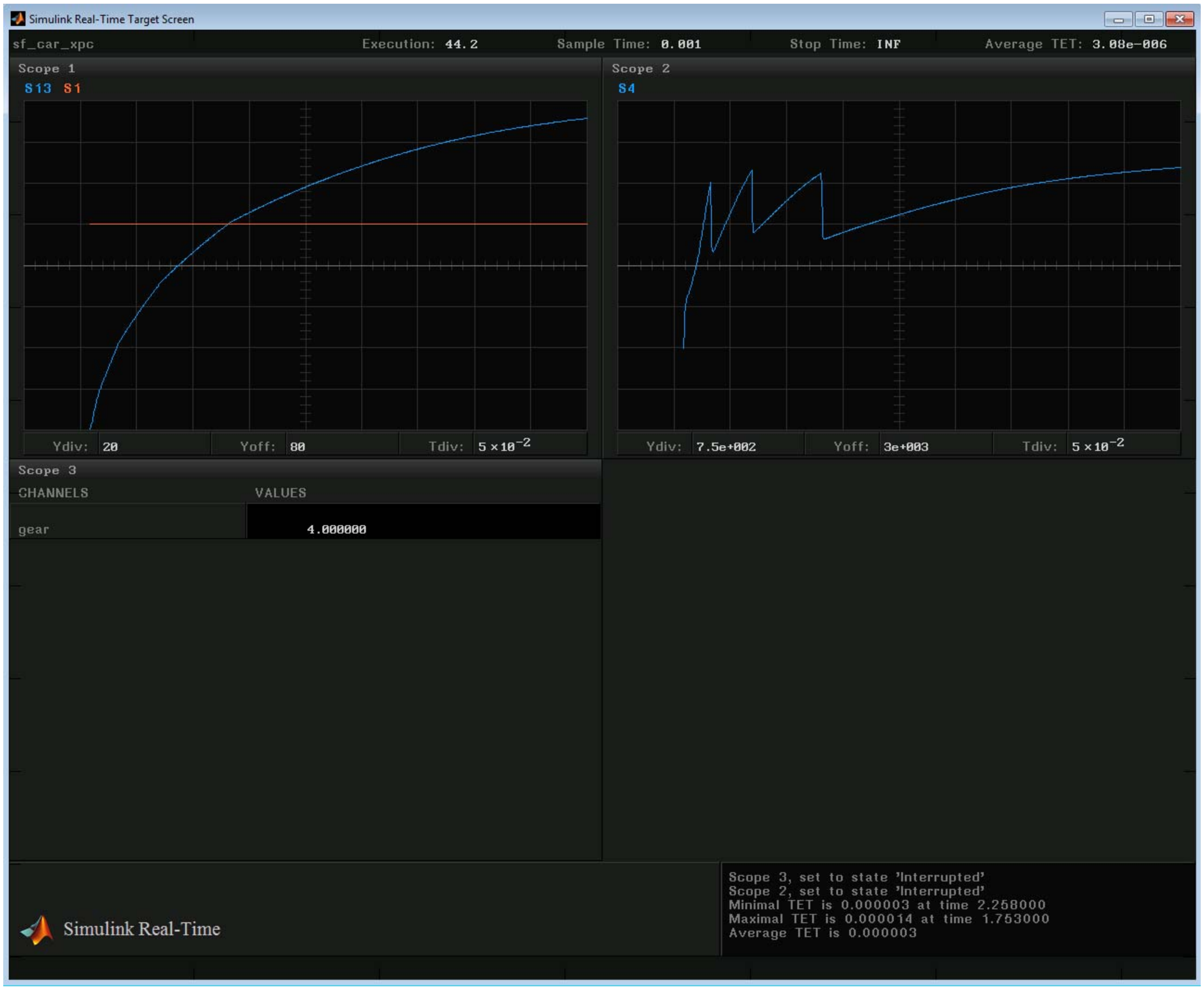
The target computer displays the following messages and three scopes.



```
Scope 2, NumSamples set to 1000
Scope 2, decimation set to 50
Scope 2, Trigger Mode is set to FreeRun
Initializing application finished.
Model thread stack is 128K bytes
```

6 Hold down the **Shift** key and hold down **T** until the value of `Th` reaches 100.

7 Press **s** to start the application.



```
sf_car_xpc          Execution: 44.2     Sample Time: 0.001     Stop Time: INF     Average TET: 3.08e-006
```

CHANNELS	VALUES
gear	4.000000

```
Scope 3, set to state 'Interrupted'
Scope 2, set to state 'Interrupted'
Minimal TET is 0.000003 at time 2.258000
Maximal TET is 0.000014 at time 1.753000
Average TET is 0.000003
```

In Scope 1, S1 shows the throttle rising to a maximum value of 100 and the vehicle speed S13 gradually increasing. In scope 2, S4 shows the vehicle RPM. Notice the changes in the vehicle RPM as the gears shift from first to fourth gear as displayed in the numerical Scope 3.

8 When you are done testing the example application, type **Q** or **Ctrl+C**.

   The example application is disconnected from the target computer, so you can reconnect to MATLAB.

C Code for sf_car_xpc.c

This section contains the C code for the `sf_car_xpc.c` application:

```
/* File:     sf_car_xpc.c
 * Abstract: Demonstrates the use of the Simulink Real-Time C-API in Human-Machine
 *           interaction. This file generates a Win32 Console application,
 *           which when invoked loads the sf_car_xpc.dlm compiled application
 *           on to the Simulink Real-Time PC.
 *
 *           To build the executable, use the Visual C/C++ project
 *           sf_car_xpc.dsp.
 *
 * Copyright 2000-2004 The MathWorks, Inc.
 */

/* Standard include files */
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <ctype.h>
#include <conio.h>
#include <windows.h>

/* Simulink Real-Time C-API specific includes */
#include "xpcapi.h"
#include "xpcapiconst.h"

#define SERIAL 0
#define TCPIP  1

/* max and min are defined by some compilers, so we wrap them in #ifndef's */
#ifndef max
#define max(a, b) (((a) > (b)) ? (a) : (b))
#endif
#ifndef min
#define min(a, b) (((a) < (b)) ? (a) : (b))
#endif

/* Global Variables */
int   mode = TCPIP, comPort = 0;
int   port;
int   thrPID, brakePID, rpmSID, speedSID, gearSID;
char *ipAddress, *ipPort, *pathToApp = NULL;

/* Function prototypes */
double getParam(int parIdx);
void   setParam(int parIdx, double parValue);
void   findParam(char *block, char *param, int *id);
void   findSignal(char *sig, int *id);

void   Usage(void);
void   cleanUp(void);
void   checkError(char *str);
void   processKeys(void);
void   parseArgs(int argc, char *argv[]);
int    str2Int(char *str);


/* Function: main ============================================================
 * Abstract: Main function for the sf_car_xpc demo                          */
int main(int argc, char *argv[]) {
    printf("\n"
           "*-------------------------------------------------------------*\n"
           "*          Simulink Real-Time API Demo: sf_car_xpc.                 *\n"
           "*                                                             *\n"
           "* Copyright (c) 2000 The MathWorks, Inc. All Rights Reserved. *\n"
           "*-------------------------------------------------------------*\n"
           "\n");

    parseArgs(argc, argv);
    atexit(cleanUp);
    /* Initialize the API */
    if (xPCInitAPI()) {
        fprintf(stderr, "Could not load api\n");
        return -1;
    }

    if (mode == SERIAL)
        port = xPCOpenSerialPort(comPort, 0);
    else if (mode == TCPIP)
```

```
        port = xPCOpenTcpIpPort(ipAddress, ipPort);
    else {
        fprintf(stderr, "Invalid communication mode\n");
        exit(EXIT_FAILURE);
    }
    checkError("PortOpen: ");

    xPCLoadApp(port, ".", "sf_car_xpc"); checkError("LoadApp: ");
    printf("Application sf_car_xpc loaded, SampleTime: %g  StopTime: %g\n\n",
           xPCGetSampleTime(port), xPCGetStopTime(port));
    checkError(NULL);

    findParam("Throttle", "Value", &thrPID);
    findParam("Brake", "Value", &brakePID);
    findSignal("Engine/rpm", &rpmSID);
    findSignal("Vehicle/mph", &speedSID);
    findSignal("shift_logic/p1", &gearSID);

    processKeys();                      /* Heart of the application */

    if (xPCIsAppRunning(port)) {
        xPCStopApp(port);
    }
    return 0;
} /* end main() */

/* Function: processKeys ====================================================
 * Abstract: This function reads and processes the keystrokes typed by the
 *           user and takes action based on them. This function runs for most
 *           of the program life.                                           */
void processKeys(void) {
    int    c = 0;
    double throttle, brake;

    throttle = getParam(thrPID);
    brake    = getParam(brakePID);
    fputs("\nR    Br    Th  G     VehSpeed     VehRPM  \n", stdout);
    fputs(  "-   ----   --  -    ----------   -------- \n", stdout);
    while (1) {
        if (_kbhit()) {
            c = _getch();
            switch (c) {
              case 't':
                if (throttle)
                    setParam(thrPID, --throttle);
                break;
              case 'T':
                if (brake)
                    setParam(brakePID, (brake = 0));
                if (throttle < 100)
                    setParam(thrPID, ++throttle);
                break;
              case 'b':
                setParam(brakePID, (brake = max(brake - 200, 0)));
                if (brake)
                    setParam(thrPID, (throttle = 0));
                break;
              case 'B':
                if (throttle)
                    setParam(thrPID, (throttle = 0));
                setParam(brakePID, (brake = min(brake + 200, 4000)));
                break;
              case 's':
              case 'S':
                if (xPCIsAppRunning(port)) {
                    xPCStopApp(port);  checkError(NULL);
                } else {
                    xPCStartApp(port); checkError(NULL);
                }
                break;
              case 'q':
              case 'Q':
                return;
                break;
              default:
                fputc(7, stderr);
                break;
            }
        } else {
            Sleep(50);
        }
        printf( "\r%c   %4d  %3d  %1d  %10.3f   %10.3f",
                (xPCIsAppRunning(port) ? 'Y' : 'N'),
                (int)brake, (int)throttle,
                (int)xPCGetSignal(port, gearSID),
                xPCGetSignal(port, speedSID),
                xPCGetSignal(port, rpmSID));
    }
} /* end processKeys() */

/* Function: Usage ===========================================================
 * Abstract: Prints a simple usage message.                                 */
void Usage(void) {
    fprintf(stdout,
            "Usage: sf_car_xpc {-t IPAddress:IpPort|-c num}\n\n"
            "E.g.:  sf_car_xpc -t 192.168.0.1:22222\n"
            "E.g.:  sf_car_xpc -c 1\n\n");
    return;
} /* end Usage() */

/* Function: str2Int =========================================================
 * Abstract: Converts the supplied string str to an integer. Returns INT_MIN
 *           if the string is invalid as an integer (e.g. "123string" is
 *           invalid) or if the string is empty.                            */
int str2Int(char *str) {
    char *tmp;
    int   tmpInt;
    tmpInt = (int)strtol(str, &tmp, 10);
    if (*str == '\0' || (*tmp != '\0')) {
        return INT_MIN;
    }
    return tmpInt;
} /* end str2Int */

/* Function: parseArgs =======================================================
 * Abstract: Parses the command line arguments and sets the state of variables
 *           based on the arguments.                                        */
void parseArgs(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "Insufficient command line arguments.\n\n");
```

```
            Usage();
            exit(EXIT_FAILURE);
        }
        if (strlen(argv[1]) != 2               ||
            strchr("-/",   argv[1][0]) == NULL ||
            strchr("tTcC", argv[1][1]) == NULL) {
            fprintf(stderr, "Unrecognized Argument %s\n\n", argv[1]);
            Usage();
            exit(EXIT_FAILURE);
        }
        mode = tolower(argv[1][1]) == 'c' ? SERIAL : TCPIP;
        if (mode == SERIAL) {
            int tmpInt;
            if ((tmpInt = str2Int(argv[2])) > INT_MIN) {
                comPort = tmpInt;
            } else {
                fprintf(stderr, "Unrecognized argument %s\n", argv[2]);
                Usage();
            }
        } else {
            char *tmp;
            ipAddress = argv[2];
            if ((tmp = strchr(argv[2], ':')) == NULL) {
                /* memory need not be freed as it is allocated only once, will *
                 * hang around till app ends.                                  */
                if ((ipPort = malloc(6 * sizeof(char))) == NULL) {
                    fprintf(stderr, "Unable to allocate memory");
                    exit(EXIT_FAILURE);
                }
                strcpy(ipPort, "22222");
            } else {
                *tmp      = '\0';
                ipPort    = ++tmp;
            }
        }
        return;
    } /* end parseArgs() */

    /* Function: cleanUp ==========================================================
     * Abstract: Called at program termination to exit in a clean way.          */
```

```
void cleanUp(void) {
    xPCClosePort(port);
    xPCFreeAPI();
    return;
} /* end cleanUp() */

/* Function: checkError ======================================================
 * Abstract: Checks for error by calling xPCGetLastError(); if an error is
 *           found, prints the error message and exits.         */
void checkError(char *str) {
    char errMsg[80];
    if (xPCGetLastError()) {
        if (str != NULL)
            fputs(str, stderr);
        xPCErrorMsg(xPCGetLastError(), errMsg);
        fputs(errMsg, stderr);
        exit(EXIT_FAILURE);
    }
    return;
} /* end checkError() */

/* Function: findParam =======================================================
 * Abstract: Wrapper function around the xPCGetParamIdx() API call. Also
 *           checks to see if the parameter is not found, and exits in that
 *           case.                                                         */
void findParam(char *block, char *param, int *id) {
    int tmp;
    tmp = xPCGetParamIdx(port, block, param);
    if (xPCGetLastError() || tmp == -1) {
        fprintf(stderr, "Param %s/%s not found\n", block, param);
        exit(EXIT_FAILURE);
    }
    *id = tmp;
    return;
} /* end findParam() */

/* Function: findSignal ======================================================
 * Abstract: Wrapper function around the xPCGetSignalIdx() API call. Also
 *           checks to see if the signal is not found, and exits in that
 *           case.                                                         */
```

```
void findSignal(char *sig, int *id) {
    int tmp;
    tmp = xPCGetSignalIdx(port, sig);
    if (xPCGetLastError() || tmp == -1) {
        fprintf(stderr, "Signal %s not found\n", sig);
        exit(EXIT_FAILURE);
    }
    *id = tmp;
    return;
} /* end findSignal() */

/* Function: getParam ==========================================================
 * Abstract: Wrapper function around the xPCGetParam() API call. Also checks
 *           for error, and exits if an error is found.                     */
double getParam(int parIdx) {
    double p;
    xPCGetParam(port, parIdx, &p);
    checkError("GetParam: ");
    return p;
} /* end getParam() */

/* Function: setParam ==========================================================
 * Abstract: Wrapper function around the xPCSetParam() API call. Also checks
 *           for error, and exits if an error is found.                     */
void setParam(int parIdx, double parValue) {
    xPCSetParam(port, parIdx, &parValue);
    checkError("SetParam: ");
    return;
} /* end setParam() */

/** EOF sf_car_xpc.c **/
```

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 target 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" on page 4-2
"Accessing the Demo Project Solution" on page 4-3
"Rebuilding the Demo Project Solution" on page 4-3
"Using the Demo Executable" on page 4-4

Before Starting

To use the `Demo` solution, you need

- A target computer running a current Simulink Real-Time kernel
- A host computer running the MATLAB software interface, connected to the target computer via RS-232 or TCP/IP
- A target 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 host 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. To deploy a GUI application to other host computer systems that do not have your licensed copy of the Simulink Real-Time product, you need the Simulink Real-Time standalone mode.

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.

Simulink Real-Time API Reference for Microsoft .NET Framework

Simulink Real-Time API for Microsoft .NET Framework — Alphabetical List

Namespace: `MathWorks.xPCTarget.FrameWork`

Purpose

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

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

Purpose

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

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.

Purpose

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

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

Purpose

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

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.

Purpose

Create `xPCTargetScope` object

Syntax

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

Description

Class: `xPCTargetScopeCollection Class`

Method

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

Purpose

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

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

Purpose

Close current stream

Syntax

`public void Close()`

Description

Class: `xPCFileStream Class`

Method

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.

Purpose

Establish connection to target computer

Syntax

```
public void Connect()
```

Description

Class: `xPCTargetPC Class`

Method

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

Purpose

Asynchronous request for target computer connection

Syntax

```
public void ConnectAsync()
```

Description

Class: `xPCTargetPC Class`

Method

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.

Purpose

Event when `xPCTargetPC.ConnectAsync` is complete

Syntax

```
public event ConnectCompleted ConnectCompleted
```

Description

Class: `xPCTargetPC Class`

Event

Syntax Language: C#

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

Purpose

Event after `xPCTargetPC.Connect` is complete

Syntax

```
public event EventHandler Connected
```

Description

Class: `xPCTargetPC Class`

Event

Syntax Language: C#

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

Purpose Event before `xPCTargetPC.Connect` starts

Syntax `public event EventHandler Connecting`

Description **Class:** `xPCTargetPC Class`

Event

Syntax Language: C#

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

xPCFileInfo.CopyToHost

Purpose

Copy file from target computer file system to host file system

Syntax

`public FileInfo CopyToHost(string HostDestFileName)`

Description

Class: `xPCFileInfo Class`

Method

Syntax Language: C#

`public FileInfo CopyToHost(string HostDestFileName)` copies file, *`HostDestFileName`*, from target computer file system to new location on host 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.
`ArgumentNull-Exception`	`HostDestFileName` is NULL reference.
`NotSupported-Exception`	*`HostDestFileName`* contains a colon (:) in the middle of the string.
`PathTooLong-Exception`	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.
`SecurityException`	Caller does not have required permission.
`UnauthorizedAccess-Exception`	System does not allow access to *`HostDestFileName`*.
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

Purpose Create file in specified path

Syntax `public xPCFileStream Create()`

Description **Class:** `xPCFileInfo Class`

Method

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

Purpose

Create folder

Syntax

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

Description

Class: `xPCFileSystem Class`

Method

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.

Purpose Create folder

Syntax `public void Create()`

Description **Class:** `xPCDirectoryInfo Class`

Method

Syntax Language: C#

`public void Create()` creates a folder.

xPCFileSystemInfo.Delete

Purpose Delete current file or folder

Syntax `public abstract void Delete()`

Description **Class:** `xPCFileSystemInfo Class`

Method

Syntax Language: C#

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

Purpose Delete empty `xPCDirectoryInfo` object

Syntax `public override void Delete()`

Description **Class:** `xPCDirectoryInfo Class`

Method

Syntax Language: C#

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

xPCFileInfo.Delete

Purpose

Permanently delete file on target computer

Syntax

`public override void Delete()`

Description

Class: `xPCFileInfo Class`

Method

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.

Purpose

Disconnect from target computer

Syntax

```
public void Disconnect()
```

Description

Class: `xPCTargetPC Class`

Method

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

Purpose

Asynchronous request to disconnect from target computer

Syntax

`public void DisconnectAsync()`

Description

Class: `xPCTargetPC Class`

Method

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.

Purpose

Event when `xPCTargetPC.DisconnectAsync` is complete

Syntax

```
public event DisconnectCompletedEventHandler DisconnectCompleted
```

Description

Class: `xPCTargetPC Class`

Event

Syntax Language: C#

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

xPCTargetPC.Disconnected

Purpose Event after `xPCTargetPC.Disconnect` is complete

Syntax `public event EventHandler Disconnected`

Description **Class:** `xPCTargetPC Class`

Event

Syntax Language: C#

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

Purpose

Event before `xPCTargetPC.Disconnect` starts

Syntax

```
public event EventHandler Disconnecting
```

Description

Class: `xPCTargetPC Class`

Event

Syntax Language: C#

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

xPCTargetPC.Dispose

Purpose

Clean up used resources

Syntax

`public void Dispose()`

Description

Class: `xPCTargetPC Class`

Method

Syntax Language: C#

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

Exception

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

Purpose

Event after `xPCTargetPC.Dispose` is complete

Syntax

```
public event EventHandler Disposed
```

Description

Class: `xPCTargetPC Class`

Event

Syntax Language: C#

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

xPCFileSystem.GetCurrentDirectory

Purpose

Current working folder for target application

Syntax

`public string GetCurrentDirectory()`

Description

Class: `xPCFileSystem Class`

Method

Syntax Language: C#

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

Exception

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

Purpose

Copy signal data from target computer

Syntax

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

Description

Class: `xPCDataLoggingObject Class`

Method

Syntax Language: C#

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

xPCDataFileScSignalObject.GetData

Purpose

Copy file scope signal data from target computer

Syntax

`public double[] GetData()`

Description

Class: `xPCDataFileScSignalObject Class`

Method

Syntax Language: C#

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

Purpose

Copy host scope signal data from target computer

Syntax

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

Description

Class: `xPCDataHostScSignalObject Class`

Method

Syntax Language: C#

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

xPCDataLoggingObject.GetDataAsync

Purpose

Asynchronously copy signal data from target computer

Syntax

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

Description

Class: `xPCDataLoggingObject Class`

Method

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.

Purpose

Asynchronously copy file scope signal data from target computer

Syntax

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

Description

Class: `xPCDataFileScSignalObject Class`

Method

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

Purpose

Asynchronously copy host scope signal data from target computer

Syntax

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

Description

Class: `xPCDataHostScSignalObject Class`

Method

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.

Purpose

Event when `xPCDataLoggingObject.GetDataAsync` is complete

Syntax

```
public event GetDataCompletedEventHandler GetDataCompleted
```

Description

Class: `xPCDataLoggingObject Class`

Event

Syntax Language: C#

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

xPCDataFileScSignalObject.GetDataCompleted

Purpose Event when `xPCDataFileScSignalObject.GetDataAsync` is complete

Syntax

```
public event GetFileScSignalDataCompletedEventHandler GetDataCompleted
```

Description

Class: `xPCDataFileScSignalObject Class`

Event

Syntax Language: C#

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

Purpose

Event when `xPCDataHostScSignalObject.GetDataAsync` is complete

Syntax

```
public event GetDataCompletedEventHandler GetDataCompleted
```

Description

Class: `xPCDataHostScSignalObject Class`

Event

Syntax Language: C#

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

xPCDirectoryInfo.GetDirectories

Purpose Subfolders of current folder

Syntax

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

Description

Class: `xPCDirectoryInfo Class`

Method

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

Purpose

Drive names for logical drives on target computer

Syntax

`public xPCDriveInfo[] GetDrives()`

Description

Class: `xPCFileSystem Class`

Method

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

Purpose File list from current folder

Syntax `public xPCFileInfo[] GetFiles()`

Description **Class:** `xPCDirectoryInfo Class`

Method

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.

Purpose

File system information for files and subfolders in folder

Syntax

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

Description

Class: `xPCDirectoryInfo Class`

Method

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

Purpose

Get parameter values from target computer

Syntax

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

Description

Class: `xPCParameter Class`

Method

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.

Purpose

Asynchronous request to get parameter values from target computer

Syntax

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

Description

Class: `xPCParameter Class`

Method

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

Purpose

Event when `xPCParameter.GetParamAsync` is complete

Description

Class: `xPCParameter Class`

Event

Syntax Language: C#

```
public event GetParamCompletedEventHandler
```

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

Purpose

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

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

Purpose

Vector of signal values from array

Syntax

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

Description

Class: `xPCSignals Class`

Method

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.

Purpose Value of signal at moment of request

Syntax

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

Description

Class: `xPCSignal Class`

Method

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

Purpose

Load target application onto target computer

Syntax

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

Description

Class: `xPCTargetPC Class`

Method

Syntax Language: C#

`public xPCApplication Load()` loads a target 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 target 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.
`InvalidOperation-Exception`	*`DLMFileName`* is a NULL reference (empty in Visual Basic) or an empty string.
`NotSupported-Exception`	*`DLMFileName`* contains a colon (:) in the middle of the string.
`PathTooLong-Exception`	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.

Exception	Condition
`SecurityException`	Caller does not have required permission.
`UnauthorizedAccess-Exception`	System does not allow access to *`DLMFileName`*.

xPCTargetPC.LoadAsync

Purpose

Asynchronous request to load target application onto target computer

Syntax

`public void LoadAsync()`

Description

Class: `xPCTargetPC Class`

Method

Syntax Language: C#

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

Exception

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

Purpose

Event when `xPCTargetPC.LoadAsync` is complete

Syntax

```
public event LoadCompletedEventHandler LoadCompleted
```

Description

Class: `xPCTargetPC Class`

Event

Syntax Language: C#

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

xPCTargetPC.Loaded

Purpose Event after `xPCTargetPC.Load` is complete

Syntax `public event EventHandler Loaded`

Description **Class:** `xPCTargetPC Class`

Event

Syntax Language: C#

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

Purpose Event before `xPCTargetPC.Load` starts

Syntax `public event EventHandler Loading`

Description **Class:** `xPCTargetPC Class`

Event

Syntax Language: C#

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

xPCParameters.LoadParameterSet

Purpose

Load parameter values for target application

Syntax

`public void LoadParameterSet(string fileName)`

Description

Class: `xPCParameters Class`

Method

Syntax Language: C#

`public void LoadParameterSet(string fileName)` loads parameter values for the target 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

Purpose `CancelPropertyNotification` event data

Syntax

```
public class CancelPropertyNotificationEventArgs : PropertyNotificatio
   nEventArgs
```

Description **Namespace:** `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class CancelPropertyNotificationEventArgs : PropertyNotificatio nEventArgs` 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

Purpose `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

Purpose

`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

Purpose

`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

Purpose

`xPCDataFileScSignalObject.GetDataCompleted` event data

Syntax

```
public class GetFileScSignalDataObjectCompletedEventArgs : GetDataComp
   letedEventArgs
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class GetFileScSignalDataObjectCompletedEventArgs : GetDataComp letedEventArgs` 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

Purpose

`xPCDataHostScSignalObject.DataObjectCompleted` event data

Syntax

```
public class GetHostScSignalDataObjectCompletedEventArgs : GetDataComp
   letedEventArgs
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public class GetHostScSignalDataObjectCompletedEventArgs : GetDataComp letedEventArgs` 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`).
`ScopeSignalObject`	`public xPCScopeSignal ScopeSignalObject {get;}`	Get reference to parent xPCHostScopeSignal object

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.

GetLogDataCompletedEventArgs Class

Purpose

`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

Purpose

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

Purpose

`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 target 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

Purpose

`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

Purpose `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.

Purpose

`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

Purpose

`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 target 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

Purpose

Access to target 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 target application execution
`xPCApplication.Stop`	Stop target 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.

Properties	C# Declaration Syntax	Description	Exception
`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 target application	`xPCException` — When problem occurs, query xPCException object `Reason` property.
`Parameters`	`public xPCParameters Parameters {get;}`	Get reference to the xPCParameters object.	
`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	

Properties	C# Declaration Syntax	Description	Exception
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.	

Purpose Access to target 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
`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.
`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

Purpose

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.GetData`	Copy file scope signal data from target computer
`xPCDataFileScSignalObject.GetDataAsync`	Asynchronously copy file scope signal data from target computer

Events

Event	Description
`xPCDataFileScSignalObject.GetDataCompleted`	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

Purpose

Object that holds logged host scope signal data

Syntax

```
public class xPCDataHostScSignalObject : xPCApplicationNotficationObje
   ct, IxPCDataService, IxPCDataServiceAsync
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Methods

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

Events

Event	Description
`xPCDataHostScSignalObject.GetDataCompleted`	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.

Purpose

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.GetData`	Copy signal data from target computer
`xPCDataLoggingObject.GetDataAsync`	Asynchronously copy signal data from target computer

Events

Event	Description
`xPCDataLoggingObject.GetDataCompleted`	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.
`LogId`	`public int LogId {get;}`	

Property	C# Declaration Syntax	Description
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

Purpose

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.GetDirectories`	Subfolders of current folder
`xPCDirectoryInfo.GetFiles`	File list from current folder
`xPCDirectoryInfo.GetFileSystemInfos`	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
`Root`	`public xPCDirectoryInfo Root {get;}`	Get the root portion of a path.	`xPCException` — When problem occurs, query xPCException object `Reason` property.

Purpose

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.
`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.

Property	C# Declaration Syntax	Description	Exception
Name	public string Name {get;}	Get name of drive.	xPCException — When problem occurs, query xPCException object Reason property.
Root-Directory	public xPCDirectoryInfo RootDirectory	Get root folder of drive.	xPCException — When problem occurs, query xPCException object
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

Purpose

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.
`Reason`	`public xPCExceptionReason Reason {get;}`	Get xPCExceptionReason reason. See `xPCExceptionReason Enumerated Data Type`.

Property	C# Declaration Syntax	Description
`Source`	`public virtual string Source {get; set;}`	Get or set name of target 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.

Purpose

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 host 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.

Property	C# Declaration Syntax	Description
`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

Purpose

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 xPCDataHostScSignalObject 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.	

Property	C# Declaration Syntax	Description	Exception
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, 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.	xPCException — When problem occurs, query xPCException object Reason property.
Signals	public xPCTarget-ScopeSignalCollection Signals {get;}	Get collection of file scope signals (xPCFileScope-SignalCollection) assigned to this scope object.	

Property	C# Declaration Syntax	Description	Exception
`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

Purpose 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.Refresh`	Synchronize with file scopes on target computer
`xPCFileScopeCollection.StartAll`	Start all file scopes in one call
`xPCFileScopeCollection.StopAll`	Stop all file scopes in one call

xPCFileScopeSignal Class

Purpose

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.

Purpose

Collection of `xPCFileScopeSignal` objects

Syntax

```
public class xPCFileScopeSignalCollection : xPCScopeSignalCollection<x
   PCFileScopeSignal>
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Methods

Method	Description
`xPCFileScopeSignalCollection.Add`	Add signals to file scope
`xPCFileScopeSignalCollection.Refresh`	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 parent xPCHostScope object. This property returns the file scope	`xPCException` — When problem occurs, query xPCException object `Reason` property.

Property	C# Declaration Syntax	Description	Exception
		signal object as type xPCFileScopeSignal.	

Purpose 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 xPCException object `Reason` property.

xPCFileSystem Class

Purpose

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.GetCurrentDirectory`	Current working folder for target application
`xPCFileSystem.GetDrives`	Drive names for the logical drives on the target computer
`xPCFileSystem.RemoveFile`	Remove file name from target computer
`xPCFileSystem.SetCurrentDirectory`	Current folder

xPCFileSystemInfo Class

Purpose 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.
`Extension`	`public string Extension {get;}`	Get string that represents file extension.

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

Purpose 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
`DataTime-Object`	`public xPCDataHostSc-SignalObject DataTimeObject {get;}`	Get host scope time data object xPCDataHost-ScSignalObject associated with this scope.	
`Signals`	`public xPCTarget-ScopeSignal-`	Get collection of host scope signals (xPCHost-	

Property	C# Declaration Syntax	Description	Exception
	`Collection Signals {get;}`	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.

Purpose

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.Refresh`	Refresh host scope object state
`xPCHostScopeCollection.StartAll`	Start all host scopes in one call
`xPCHostScopeCollection.StopAll`	Stop all host scopes in one call

xPCHostScopeSignal Class

Purpose 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

Purpose 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
`xPCHostScopeSignalCollection.Add`	Create xPCHostScopeSignal object
`xPCHostScopeSignalCollection.Refresh`	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 parent xPCHostScope object. This property returns the file scope signal	`xPCException` — When problem occurs, query xPCException object `Reason` property.

Property	C# Declaration Syntax	Description	Exception
		object as type xPCHostScopeSignal.	

Purpose

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

Purpose

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.
`NumOutputs`	`public int NumOutputs {get;}`	Return a reference to the xPCOutputLogger object.

xPCParameter Class

Purpose

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 target application parameter. You can tune the parameter using xPCParameter objects.

Methods

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

Events

Event	Description
`xPCParameter.GetParamCompleted`	Event when `xPCParameter.GetParamAsync` is complete
`xPCParameter.SetParamCompleted`	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	
`Parameter-Id`	`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

Purpose

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.LoadParameterSet`	Load parameter values for target application
`xPCParameters.Refresh`	Refresh state of object
`xPCParameters.SaveParameterSet`	Save parameter values of target application

Properties

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

Property	C# Declaration Syntax	Description
		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

Purpose

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.

Property	C# Declaration Syntax	Description	Exception
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.	xPCException — When problem occurs, query xPCException object Reason property.
ScopeId	public int ScopeId {get;}	A numeric index, unique for each scope.	

Property	C# Declaration Syntax	Description	Exception
`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.
`TriggerAnySignal`	`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.
`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.

Property	C# Declaration Syntax	Description	Exception
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.EITHER (default), SLTRIGGERSLOPE.RISING, and SLTRIGGERSLOPE.FALLING. 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.	

Purpose

`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

Purpose

`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

Purpose

`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

Purpose 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).
`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

Purpose

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.
`SignalId`	`public virtual int SignalId {get;}`	Get numeric identifier that represents the signal object.

Property	C# Declaration Syntax	Description
`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

Purpose

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
`NumSignals`	`public int NumSignals {get;}`	Get total numbers of signals available in target application.	
`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.	`xPCException` — When problem occurs, query xPCException object `Reason` property. `ArgumentNullException` — *`signalIdx`* or

Property	C# Declaration Syntax	Description	Exception
		blkPath is a string that specifies the block path name for the signal.	*blkPath* is NULL reference.

Purpose

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

Purpose 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.DisconnectAsync`	Asynchronous request to disconnect from target computer
`xPCTargetPC.Dispose`	Clean up used resources
`xPCTargetPC.Load`	Load target application onto target computer
`xPCTargetPC.LoadAsync`	Asynchronous request to load target application onto target computer
`xPCTargetPC.Ping`	Test communication between host 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

Method	Description
`xPCTargetPC.Unload`	Unload target application from target computer
`xPCTargetPC.UnloadAsync`	Asynchronous request to unload target application from target computer

Events

Event	Description
`xPCTargetPC.ConnectCompleted`	Event when `xPCTargetPC.ConnectAsync` is complete
`xPCTargetPC.Connected`	Event after `xPCTargetPC.Connect` is complete
`xPCTargetPC.Connecting`	Event before `xPCTargetPC.Connect` starts
`xPCTargetPC.DisconnectCompleted`	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.RebootCompleted`	Event when `xPCTargetPC.RebootAsync` is complete
`xPCTargetPC.Rebooted`	Event after `xPCTargetPC.Reboot` is complete
`xPCTargetPC.Rebooting`	Event before `xPCTargetPC.Reboot` starts
`xPCTargetPC.UnloadCompleted`	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 target 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.	

Property	C# Declaration Syntax	Description	Exception
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.
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).	

Property	C# Declaration Syntax	Description	Exception
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 connection. See xPCRS232BaudRate Enumerated Data Type.	
RS232HostPort	public XPCRS232CommPort RS232HostPort {get; set;}	Get or set the serial COM port for connection on host computer. The Simulink Real-Time software automatically determines the COM port on the target computer. See xPCRS232Comport Enumerated Data Type.	

Property	C# Declaration Syntax	Description	Exception
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

Purpose

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
`Display-Mode`	`public SCDISPLAYMODE DisplayMode {get; set;}`	Get or set scope mode for displaying signals.	`xPCException` — When problem occurs, query xPCException object `Reason` property.
`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.

Property	C# Declaration Syntax	Description	Exception
`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

Purpose Collection of `xPCTargetScope` objects

Syntax

```
public class xPCTargetScopeCollection : xPCScopeCollection<xPCTargetSc
   ope>
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Methods

Method	Description
`xPCTargetScopeCollection.Add`	Create xPCTargetScope object with the next available scope ID as key
`xPCTargetScopeCollection.Refresh`	Refresh target scope object state
`xPCTargetScopeCollection.StartAll`	Start all target scopes in one call
`xPCTargetScopeCollection.StopAll`	Stop all target scopes in one call

xPCTargetScopeSignalCollection Class

Purpose Collection of `xPCHostScopeSignal` objects

Syntax

```
public class xPCTargetScopeSignalCollection : xPCScopeSignalCollectio
```

Description **Namespace:** `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Methods

Method	Description
`xPCTargetScopeSignalCollection.Add`	Create xPCTargetScopeSignal object
`xPCTargetScopeSignalCollection.Refresh`	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 signal object added to its parent xPCTargetScope object. This property returns the file scope signal	`xPCException` — When problem occurs, query xPCException object `Reason` property.

Property	C# Declaration Syntax	Description	Exception
		object as type xPCTgtScopeSignal.	

Purpose

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

Purpose

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.	

Purpose

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

Purpose Open file

Syntax `public xPCFileStream Open(xPCFileMode fileMode)`

Description **Class:** `xPCFileInfo Class`

Method

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.

Purpose

Create read-only `xPCFileStream` object

Syntax

```
public xPCFileStream OpenRead()
```

Description

Class: `xPCFileInfo Class`

Method

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

Purpose Test communication between host and target computers

Syntax `public bool Ping()`

Description **Class:** `xPCTargetPC Class`

Method

Syntax Language: C#

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

Purpose

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

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

Purpose Restart target computer

Syntax `public void Reboot()`

Description **Class:** `xPCTargetPC Class`

Method

Syntax Language: C#

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

Exception

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

Purpose

Asynchronous request to restart target computer

Syntax

`public void RebootAsync()`

Description

Class: `xPCTargetPC Class`

Method

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

Purpose Event when `xPCTargetPC.RebootAsync` is complete

Syntax `public event RebootCompletedEventHandler RebootCompleted`

Description **Class:** `xPCTargetPC Class`

Event

Syntax Language: C#

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

Purpose Event after `xPCTargetPC.Reboot` is complete

Syntax

```
public event EventHandler Rebooted
```

Description

Class: `xPCTargetPC Class`

Event

Syntax Language: C#

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

xPCTargetPC.Rebooting

Purpose Event before `xPCTargetPC.Reboot` starts

Syntax

```
public event EventHandler Rebooting
```

Description

Class: `xPCTargetPC Class`

Event

Syntax Language: C#

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

Purpose

Synchronize with file scopes on target computer

Syntax

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

Description

Class: `xPCFileScopeCollection Class`

Method

Syntax Language: C#

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

Overrides xPCScopeCollection<xPCFileScope>.Refresh().

xPCScopes.RefreshAll

Purpose

Refresh state of object

Syntax

`public void RefreshAll()`

Description

Class: `xPCScopes Class`

Method

Syntax Language: C#

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

Purpose

Synchronize with file drives on target computer

Syntax

```
public void Refresh()
```

Description

Class: `xPCDriveInfo Class`

Method

Syntax Language: C#

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

xPCFileScopeSignalCollection.Refresh

Purpose

Synchronize with signals for associated scope on target computer

Syntax

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

Description

Class: `xPCFileScopeSignalCollection Class`

Method

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.

Purpose

Refresh host scope object state

Syntax

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

Description

Class: `xPCHostScopeCollection Class`

Method

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

Purpose

Synchronize signals for associated host scopes on target computer

Syntax

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

Description

Class: `xPCHostScopeSignalCollection Class`

Method

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.

Purpose

Refresh state of object

Syntax

`public override void Refresh()`

Description

Class: `xPCParameters Class`

Method

Syntax Language: C#

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

xPCSignals.Refresh

Purpose Refresh state of object

Syntax `public void Refresh()`

Description **Class:** `xPCSignals Class`

Method

Syntax Language: C#

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

Purpose

Refresh target scope object state

Syntax

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

Description

Class: `xPCTargetScopeCollection Class`

Method

Syntax Language: C#

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

Overrides xPCScopeCollection<xPCTargetScope>.Refresh().

xPCTargetScopeSignalCollection.Refresh

Purpose

Synchronize signals for associated target scopes on target computer

Syntax

`public override void Refresh()`

Description

Class: `xPCTargetScopeSignalCollection Class`

Method

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.

Purpose

Remove file name from target computer

Syntax

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

Description

Class: `xPCFileSystem Class`

Method

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

Purpose Rename file

Syntax `public xPCFileInfo Rename(string newName)`

Description **Class:** `xPCFileInfo Class`

Method

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.

Purpose

Save parameter values of target application

Syntax

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

Description

Class: `xPCParameters Class`

Method

Syntax Language: C#

`public void SaveParameterSet(string fileName)` saves parameter values of the target 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

Purpose Target scope display mode values

Syntax `public enum SCDISPLAYMODE`

Description **Enumerated Data Type**

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

Purpose

Write mode values for when file allocation table entry is updated

Syntax

`public enum SCFILEMODE`

Description

Enumerated Data Type

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

Purpose

Event after `xPCScope.Start` is complete

Syntax

```
public event EventHandler ScopeStarted
```

Description

Class: `xPCScope Class`

Event

Syntax Language: C#

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

Purpose

Event before `xPCScope.Start` executes

Syntax

```
public event EventHandler ScopeStarting
```

Description

Class: `xPCScope Class`

Event

Syntax Language: C#

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

xPCScope.ScopeStopped

Purpose Event after `xPCScope.Stop` is complete

Syntax `public event EventHandler ScopeStarting`

Description **Class:** `xPCScope Class`

Event

Syntax Language: C#

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

Purpose Event before `xPCScope.Stop` executes

Syntax `public event EventHandler ScopeStopping`

Description **Class:** `xPCScope Class`

Event

Syntax Language: C#

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

SCSTATUS Enumerated Data Type

Purpose Scope status values

Syntax `public enum SCSTATUS`

Description **Enumerated Data Type**

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

Purpose

Scope trigger mode values

Syntax

`public enum SCTRIGGERMODE`

Description

Enumerated Data Type

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

Purpose Scope trigger slope values

Syntax `public enum SCTRIGGERSLOPE`

Description **Enumerated Data Type**

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.

Purpose Scope type

Syntax `public enum SCTYPE`

Description **Enumerated Data Type**

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

Purpose Current folder

Syntax `public void SetCurrentDirectory(string path)`

Description **Class:** `xPCFileSystem Class`

Method

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.

Purpose

Change value of parameter on target computer

Syntax

`public void SetParam(double[] values)`

Description

Class: `xPCParameter Class`

Method

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

Purpose

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

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.

Purpose

Event when `xPCParameter.SetParamAsync` is complete

Description

Class: `xPCParameter Class`

Event

Syntax Language: C#

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

xPCApplication.Start

Purpose Start target application execution

Syntax `public void Start()`

Description **Class:** `xPCApplication Class`

Method

Syntax Language: C#

`public void Start()` starts the target application simulation.

Exception

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

Purpose

Start all file scopes in one call

Syntax

```
public void StartAll()
```

Description

Class: `xPCFileScopeCollection Class`

Method

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

Purpose

Start all host scopes in one call

Syntax

```
public void StartAll()
```

Description

Class: `xPCHostScopeCollection Class`

Method

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.

Purpose Start all target scopes in one call

Syntax `public void StartAll()`

Description **Class:** `xPCTargetScopeCollection Class`

Method

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

Purpose

Start scope

Syntax

`public void Start()`

Description

Class: `xPCScope Class`

Method

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.

Purpose Event after `xPCApplication.Start` is complete

Syntax

```
public event EventHandler Started
```

Description

Class: `xPCApplication Class`

Event

Syntax Language: C#

`public event EventHandler Started` occurs after a target application start command is complete.

xPCApplication.Starting

Purpose Event before `xPCApplication.Start` executes

Syntax `public event EventHandler Starting`

Description **Class:** `xPCApplication Class`

Event

Syntax Language: C#

`public event EventHandler Starting` occurs before a target application start command executes.

Purpose

Stop target application execution

Syntax

```
public void Stop()
```

Description

Class: `xPCApplication Class`

Method

Syntax Language: C#

`public void Stop()` stops the target application simulation.

Exception

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

xPCFileScopeCollection.StopAll

Purpose Stop all file scopes in one call

Syntax `public void StopAll()`

Description **Class:** `xPCFileScopeCollection Class`

Method

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

Purpose

Stop all host scopes in one call

Syntax

```
public void StopAll()
```

Description

Class: `xPCHostScopeCollection Class`

Method

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

Purpose Stop all target scopes in one call

Syntax `public void StopAll()`

Description **Class:** `xPCTargetScopeCollection Class`

Method

Syntax Language: C#

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

Purpose

Stop scope

Syntax

`public void Stop()`

Description

Class: `xPCScope Class`

Method

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

Purpose Event after `xPCApplication.Stop` is complete

Syntax `public event EventHandler Stopped`

Description **Class:** `xPCApplication Class`

Event

Syntax Language: C#

`public event EventHandler Stopped` occurs after a target application stop command is complete.

Purpose Event before `xPCApplication.Stop` executes

Syntax `public event EventHandler Stopping`

Description **Class:** `xPCApplication Class`

Event

Syntax Language: C#

`public event EventHandler Stopping` occurs before a target application stop command executes.

xPCTargetPC.tcpPing

Purpose

Determine TCP/IP accessibility of remote computer

Syntax

```
public bool tcpPing()
```

Description

Class: `xPCTargetPC Class`

Method

Syntax Language: C#

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

Purpose

Software-trigger start of data acquisition for scope

Syntax

```
public void Trigger()
```

Description

Class: `xPCScope Class`

Method

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.

Purpose

Status of get signal value at moment of request

Syntax

`public virtual bool TryGetValue(ref double result)`

Description

Class: `xPCSignal Class`

Method

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.

Purpose Unload target application from target computer

Syntax `public void Unload()`

Description **Class:** `xPCTargetPC Class`

Method

Syntax Language: C#

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

Exception

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

xPCTargetPC.UnloadAsync

Purpose

Asynchronous request to unload target application from target computer

Syntax

`public void UnloadAsync()`

Description

Class: `xPCTargetPC Class`

Method

Syntax Language: C#

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

Exception

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

Purpose

Event when `xPCTargetPC.UnloadAsync` is complete

Syntax

```
public event UnloadCompletedEventHandler UnloadCompleted
```

Description

Class: `xPCTargetPC Class`

Event

Syntax Language: C#

`public event UnloadCompletedEventHandler UnloadCompleted` occurs when an asynchronous target application unload operation is complete.

Purpose Event after `xPCTargetPC.Unload` is complete

Syntax `public event EventHandler Unloaded`

Description **Class:** `xPCTargetPC Class`

Event

Syntax Language: C#

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

Purpose Event before `xPCTargetPC.Unload` starts

Syntax

```
public event EventHandler Unloading
```

Description

Class: `xPCTargetPC Class`

Event

Syntax Language: C#

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

xPCFileStream.Write

Purpose

Write block of bytes to file stream

Syntax

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

Description

Class: `xPCFileStream Class`

Method

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.

Purpose

Write byte to current position in file stream

Syntax

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

Description

Class: `xPCFileStream Class`

Method

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

Purpose Target application status return values

Syntax `public enum xPCAppStatus`

Description **Enumerated Data Type**

Syntax Language: C#

`public enum xPCAppStatus` specifies target application status return values.

Members

Member	Description
`Stopped`	Target application is stopped
`Running`	Target application is running

Purpose Construct new instance of `xPCDirectoryInfo` class on specified path

Syntax

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

Description

Class: `xPCDirectoryInfo Class`

Constructor

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

Purpose Construct new instance of `xPCDriveInfo` class

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

Description **Class:** `xPCDriveInfo Class`

Constructor

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.

Purpose

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

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

Purpose Exception reasons

Syntax `public enum xPCExceptionReason`

Description **Enumerated Data Type**

Syntax Language: C#

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

Purpose

Construct new instance of `xPCFileInfo` class

Syntax

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

Description

Class: `xPCFileInfo Class`

Constructor

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

Purpose Open file with permissions

Syntax `public enum xPCFileMode`

Description **Enumerated Data Type**

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

Purpose

Construct new instance of `xPCFileStream` class

Syntax

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

Description

Class: `xPCFileStream Class`

Method

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

Purpose Construct new instance of `xPCFileSystemInfo` class

Syntax

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

Description

Class: `xPCFileSystemInfo Class`

Constructor

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

Purpose Specify log mode values

Syntax `public enum xPCLogMode`

Description **Enumerated Data Type**

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

Purpose

Logging type values

Syntax

`public enum xPCLogType`

Description

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

Purpose

Host computer and target computer communication medium

Syntax

`public enum XPCProtocol`

Description

Enumerated Data Type

Syntax Language: C#

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

Members

Member	Description
`RS232`	Serial communication
`TCPIP`	TCP/IP communication

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

xPCRS232BaudRate Enumerated Data Type

Purpose

Serial communication baud rate

Syntax

```
public enum XPCRS232BaudRate
```

Description

Enumerated Data Type

Syntax Language: C#

`public enum XPCRS232BaudRate` specifies serial communication 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 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.

xPCRS232Comport Enumerated Data Type

Purpose Serial communication port

Syntax `public enum XPCRS232CommPort`

Description **Enumerated Data Type**

Syntax Language: C#

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

Members

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

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

xPCTargetPC

Purpose Construct new instance of `xPCTargetPC` class

Syntax `public xPCTargetPC()`

Description **Class:** `xPCTargetPC Class`

Constructor

Syntax Language: C#

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

Simulink Real-Time API Reference for C

- “C API Error Messages” on page 6-2
- “C API Structures and Functions — Alphabetical List” on page 6-6

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 link
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

Message	Description
EINVNUMPARAMS	Invalid number of parameters
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

Message	Description
EPARNOTFOUND	Parameter not found
EPARSIZMISMATCH	Parameter Size mismatch
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

Message	Description
ETGTMEMALLOC	Target memory allocation failed
ETIMELOGDISABLED	Time Logging is disabled
ETOOMANYSAMPLES	Too Many Samples requested
ETOOMANYSCOPES	Too many scopes are present
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

C API Structures and Functions — Alphabetical List

Purpose

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`

Purpose

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

Purpose

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`

Purpose

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

Purpose

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 host 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 `N`th 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`

Purpose

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

Purpose

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 target application.

Description

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

See Also

API functions `xPCMaximumTET`, `xPCMinimumTET`

Property `AvgTET` of `SimulinkRealTime.target`

Purpose

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

See Also

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

xPCClosePort

Purpose

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 Host-Target communication mode 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`

Purpose

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

Purpose

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`

Purpose

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`

Purpose

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`

Purpose

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`

Purpose

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`

Purpose

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

Purpose

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`

Purpose

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

Purpose

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`

Purpose

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

Purpose

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.

Purpose

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`

Purpose

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`

Purpose

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

Purpose

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`

Purpose

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

Purpose 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`

Purpose

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

Purpose 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`

Purpose

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

Purpose

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`

Purpose

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

Purpose

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`

Purpose

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

Purpose 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`

Purpose 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

Purpose Return target 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 target 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 target application.

Description The `xPCGetAppName` function returns the name of the target 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`

Purpose

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 target application, changes to parameters, and changes to scope signals.

See Also

API function `xPCSetEcho`

xPCGetExecTime

Purpose Return target 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 target application. If the function detects an error, it returns `-1`.

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

See Also API functions `xPCSetStopTime`, `xPCGetStopTime`

Property `ExecTime` of `SimulinkRealTime.target`

Purpose 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

Purpose

Return timeout value for communication between host computer and target computer

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 host computer and target 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 host computer and the target application. When an Simulink Real-Time API function initiates communication between the host computer and target computer, 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 target application onto the target computer, the function `xPCLoadApp` waits for a certain amount of time before checking to see if the initialization of the target application is complete. In the case where initialization of the target 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

Purpose

Return logging mode and increment value for target 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 target 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`

Purpose 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 target application. If the function detects an error, it returns `-1`.

Description The `xPCGetNumOutputs` function returns the number of outputs in the target 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

Purpose

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 target application. If the function detects an error, it returns `-1`.

Description

The `xPCGetNumParams` function returns the number of tunable parameters in the target 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`

Purpose

Return number of scopes added to target 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 target 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 target application.

xPCGetNumScSignals

Purpose 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`*.

Purpose

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 target application. If the function detects an error, it returns `-1`.

Description

The `xPCGetNumSignals` function returns the total number of signals in the target application that can be monitored from the host. 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

Purpose 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 target application. If the function detects an error, it returns `-1`.

Description The `xPCGetNumStates` function returns the number of states in the target application.

See Also API functions `xPCGetStateLog`, `xPCGetNumOutputs`, `xPCGetOutputLog`

Property `StateLog` of `SimulinkRealTime.target`

Purpose

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 target 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`

Purpose

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

Purpose

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`

Purpose

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 target 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

Purpose

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`

Purpose

Return target 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 target application. If the function detects an error, it returns `-1`.

Description

The `xPCGetSampleTime` function returns the sample time, in seconds, of the target application. You can get the error by using the function `xPCGetLastError`.

See Also

API function `xPCSetSampleTime`

Property `SampleTime` of `SimulinkRealTime.target`

Purpose

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`

Purpose

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

Purpose

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`

Purpose

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.

Purpose

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`

Purpose

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

Purpose

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`

Purpose

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

Purpose

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`

Purpose

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

Purpose 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

Purpose 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`

Purpose

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 target 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`

Purpose

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 target 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 target application. This is the amount of time the target 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

Purpose 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`

Purpose

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 target 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

Purpose

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 target 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`

Purpose

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

Purpose

Return target 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 target application is stopped, the `xPCIsAppRunning` function returns `0`. If the target 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 target 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`

Purpose

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 target 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 target application has overloaded the target computer and returns `1` if it has and `0` if it has not. If the target application is not running, the function returns `0`.

See Also

API function `xPCIsAppRunning`

Property `CPUoverload` of `SimulinkRealTime.target`

xPCIsScFinished

Purpose

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`

Purpose

Load target 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 target application file, excluding the file name. For example, in C, use a string like `"C:\\work"`.
filename	Enter the name of a compiled target application (`*.dlm`) without the file extension. For example, in C use a string like `"xpcosc"`.

Description

The `xPCLoadApp` function loads the compiled target 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 target 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`

Purpose

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 target 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

Purpose 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`

Purpose

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 target 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`

Purpose

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 target 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`

Purpose

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 target application must be stopped before you get the number.

See Also

API functions `xPCGetStateLog`, `xPCGetOutputLog`, `xPCGetTETLog`, `xPCGetTimeLog`, `xPCMaxLogSamples`

xPCNumLogWraps

Purpose 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`

Purpose 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

Purpose

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 an 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 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.

See Also

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

Purpose

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 an 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

Purpose 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`

Purpose

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

Purpose

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 host and the target. **Note** RS-232 Host-Target communication mode 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

Purpose 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`

Purpose

Save parameter values of target 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 target 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 target 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 target application parameter values from a number of application runs.

See Also

API function `xPCLoadParamSet`

xPCScAddSignal

Purpose

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`

Purpose 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

Purpose

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

Purpose

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 `N`th 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`

Purpose

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

Purpose

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`

Purpose

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`

Purpose

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.

Purpose

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`

Purpose

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`

Purpose 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.
`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.

Constant	Value	Description
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`

Purpose

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`

Purpose

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.

Constant	Value	Description
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 `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

Purpose

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

Purpose

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

Purpose

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`

Purpose 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`

Purpose 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 host 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

Purpose

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

Purpose

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

Purpose

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`

Purpose

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

Purpose

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`

Purpose

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

Purpose

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

Purpose

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`

Purpose

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

Purpose

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`

Purpose

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

Purpose

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

Purpose

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`

Purpose

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

Purpose 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 target 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`

Purpose

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

Purpose

Change initialization timeout value between host computer and target computer

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 host computer and target computer. The *`timeOut`* value is the time an Simulink Real-Time API function waits for the communication between the host computer and target computer 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 target 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`

Purpose

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

Purpose

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`

Purpose

Change target 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 target application.

Description

The `xPCSetSampleTime` function sets the sample time, in seconds, of the target application to *ts*. Use this function only when the application is stopped.

See Also

API function `xPCGetSampleTime`

Property `SampleTime` of `SimulinkRealTime.target`

xPCSetScope

Purpose

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`

Scope object methods `SimulinkRealTime.fileScope.set`, `SimulinkRealTime.hostScope.set`, and `SimulinkRealTime.targetScope.set`

Purpose

Change target 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 target application to the value in *`tfinal`*. The target 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`

Purpose

Start target 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 target application loaded on the target computer.

See Also

API function `xPCStopApp`

Target object method `SimulinkRealTime.target.start`

Purpose

Stop target 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 target application loaded on the target computer. The target 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`

Purpose

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 your host side program crashes).

See Also

API functions `xPCOpenConnection`, `xPCOpenSerialPort`, `xPCOpenTcpIpPort`, `xPCClosePort`

Purpose

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

Purpose

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

Purpose

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

Purpose

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`

Purpose

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

Purpose

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`

Purpose

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

Purpose

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`

Purpose

Unload target 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 target application, removes it from the target computer memory, and resets the target computer in preparation for receiving a new target application. The function `xPCLoadApp` calls this function before loading a new target application.

See Also

API function `xPCLoadApp`

Target object methods `SimulinkRealTime.target.load`, `SimulinkRealTime.target.unload`

Simulink Real-Time API Reference for COM

COM API Methods — Alphabetical List

Purpose

Type definition for file system folder information structure

Syntax

```
typedef struct {
BSTR Name;
BSTR Date;
BSTR Time;
long Bytes;
long isdir;
} FSDir;
```

Fields

Name	This value contains the name of the file or folder.
Date	This value contains the date the file or folder was last modified.
Time	This value contains the time the file or folder was last modified.
Bytes	This value contains the size of the file in bytes. If the element is a folder, this value is `0`.
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`.

Description

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

See Also

API method`xPCFileSystem.DirList`

FSDiskInfo

Purpose

Type definition for file system disk information structure

Syntax

```
typedef struct {
   BSTR Label;
   BSTR DriveLetter;
   BSTR Reserved;
   long SerialNumber;
   long FirstPhysicalSector;
   long FATType;
   long FATCount;
   long MaxDirEntries;
   long BytesPerSector;
   long SectorsPerCluster;
   long TotalClusters;
   long BadClusters;
   long FreeClusters;
   long Files;
   long FileChains;
   long FreeChains;
   long LargestFreeChain;
} FSDiskInfo;
```

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 address (LBA) 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 `FSDiskInfo` structure contains information for file system disks.

See Also

API method `xPCFileSystem.GetDiskInfo`

xPCFileSystem.CD

Purpose

Change current folder on target computer to specified path

Prototype

```
long CD(BSTR dir);
```

Member Of

`XPCAPICOMLib.xPCFileSystem`

Arguments

`[in]` *`dir`*	Enter the path on the target computer to change to.

Return

If the method detects an error, it returns `-1`. Otherwise, the method returns `0`.

Description

The `xPCFileSystem.CD` method changes the current folder on the target computer to the path specified in *`dir`*. Use the `xPCFileSystem.PWD` method to show the current folder of the target computer.

See Also

API method `xPCFileSystem.PWD`

xPCFileSystem.CloseFile

Purpose

Close file on target computer

Prototype

`CloseFile(long *filehandle*);`

Member Of

`XPCAPICOMLib.xPCFileSystem`

Arguments

`[in]` *`filehandle`*	Enter the file handle of an open file on the target computer.

Return

If the method detects an error, it returns `-1`. Otherwise, the method returns `0`.

Description

The `xPCFileSystem.CloseFile` method closes the file associated with *`fileHandle`* on the target computer. *`fileHandle`* is the handle of a file previously opened by the `xPCFileSystem.OpenFile` method.

See Also

API methods `xPCFileSystem.OpenFile`, `xPCFileSystem.ReadFile`, `xPCFileSystem.WriteFile`

Purpose

Return contents of target computer folder

Prototype

`DirList(BSTR *path*);`

Member Of

`XPCAPICOMLib.xPCFileSystem`

Arguments

`[in]` *`path`*	Enter the path of the folder.

Description

The `xPCFileSystem.DirList` method returns the contents of the target computer folder specified by *`path`* as an array of the `FSDir` structure.

See Also

API structure `FSDir`

API method `xPCFileSystem.GetDiskInfo`

xPCFileSystem.GetDiskInfo

Purpose Return disk information

Prototype `GetDiskInfo(BSTR driveLetter);`

Member Of `XPCAPICOMLib.xPCFileSystem`

Arguments

`[in]` *`driveLetter`*	Enter the driver letter that contains the file system.

Description The `xPCFileSystem.GetDiskInfo` method accepts as input the drive specified by *`driveLetter`* and fills in the fields of the `FSDiskInfo` structure.

See Also API structure `FSDiskInfo`

API method `xPCFileSystem.DirList`

Purpose

Return size of file on target computer

Prototype

`long GetFileSize(long *filehandle*);`

Member Of

`XPCAPICOMLib.xPCFileSystem`

Arguments

`[in]` *`filehandle`*	Enter the file handle of an open file on the target computer.

Return

This method returns the size of the specified file in bytes.

Description

The `xPCFileSystem.GetFileSize` method 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 `xPCFileSystem.OpenFile` method.

See Also

API methods `xPCFileSystem.OpenFile`, `xPCFileSystem.ReadFile`

xPCFileSystem.Init

Purpose Initialize file system object to communicate with target computer

Prototype `long Init(IxPCProtocol* xPCProtocol);`

Member Of `XPCAPICOMLib.xPCFileSystem`

Arguments

[in] `xPCProtocol`	Specify the communication port of the target computer object for which the file system is to be initialized.

Return If the method detects an error, it returns `-1`. Otherwise, the `xPCFileSystem.Init` method returns `0`.

Description The `xPCFileSystem.Init` method initializes the file system object to communicate with the target computer referenced by the `xPCProtocol` object.

Purpose Create folder on target computer

Prototype `long MKDIR(BSTR dirname);`

Member Of `XPCAPICOMLib.xPCFileSystem`

Arguments

`[in]` *`dirname`*	Enter the name of the folder to create on the target computer.

Return If the method detects an error, it returns `-1`. Otherwise, the method returns `0`.

Description The `xPCFileSystem.MKDIR` method creates the folder *`dirname`* in the current folder of the target computer.

See Also API method `xPCFileSystem.PWD`

xPCFileSystem.OpenFile

Purpose Open file on target computer

Prototype `long OpenFile(BSTR filename, BSTR permission);`

Member Of `XPCAPICOMLib.xPCFileSystem`

Arguments

`[in]` *`filename`*	Enter the name of the file to open on the target computer.
`[in]` *`permission`*	Enter the read/write permission with which to open the file. Values are `r` (read) or `w` (read/write).

Return The `xPCFileSystem.OpenFile` method returns the file handle for the opened file.

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

Note Opening the file for write access overwrites the existing contents of the file. It does not append the new data.

See Also API methods `xPCFileSystem.CloseFile`, `xPCFileSystem.GetFileSize`, `xPCFileSystem.ReadFile`, `xPCFileSystem.WriteFile`

Purpose Get current folder of target computer

Prototype `BSTR PWD();`

Member Of `XPCAPICOMLib.xPCFileSystem`

Return This method returns the path of the current folder on the target computer.

Description The `xPCFileSystem.PWD` method places the path of the current folder on the target computer.

See Also API method `xPCFileSystem.CD`

xPCFileSystem.ReadFile

Purpose Read open file on target computer

Prototype `VARIANT ReadFile(int fileHandle, int start, int numbytes);`

Member Of `XPCAPICOMLib.xPCFileSystem`

Arguments

`[in]` *`fileHandle`*	Enter the file handle of an open file on the target computer.
`[in]` *`start`*	Enter an offset from the beginning of the file from which this method can start to read.
`[in]` *`numbytes`*	Enter the number of bytes this method is to read from the file.

Return This method returns the results of the read operation as a `VARIANT` of type `Byte`. If the method detects an error, it returns `VT_ERROR`, whose value is `10`, instead.

Description The `xPCFileSystem.ReadFile` method reads an open file on the target computer and returns the results of the read operation as a `VARIANT` of type `Byte`. *`fileHandle`* is the file handle of a file previously opened by `xPCFileSystem.OpenFile`. 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 `xPCFileSystem.ReadFile` method is to read from the file.

See Also API methods `xPCFileSystem.CloseFile`, `xPCFileSystem.GetFileSize`, `xPCFileSystem.OpenFile`, `xPCFileSystem.WriteFile`

Purpose

Remove file from target computer

Prototype

`long RemoveFile(BSTR *filename*);`

Member Of

`XPCAPICOMLib.xPCFileSystem`

Arguments

`[in]` *`filename`*	Enter the name of a file on the target computer.

Return

If the method detects an error, it returns `-1`. Otherwise, the method returns `0`.

Description

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

Purpose Remove folder from target computer

Prototype `long RMDIR(BSTR dirname);`

Member Of `XPCAPICOMLib.xPCFileSystem`

Arguments

`[in]` *`dirname`*	Enter the name of a folder on the target computer.

Return If the method detects an error, it returns `-1`. Otherwise, the method returns `0`.

Description The `xPCFileSystem.RMDIR` method removes a folder named *`dirname`* from the target computer file system. *`dirname`* can be a relative or absolute path name on the target computer.

Purpose

Get name of file for scope

Prototype

`BSTR ScGetFileName(long scNum);`

Member Of

`XPCAPICOMLib.xPCFileSystem`

Arguments

`[in] scNum`	Enter the scope number.

Return

Returns the name of the file for the scope.

Description

The `xPCFileSystem.ScGetFileName` method returns the name of the file to which scope *`scNum`* will save signal data.

See Also

API method `xPCFileSystem.ScSetFileName`

Purpose Get write mode of file for scope

Prototype `long ScGetWriteMode(long scNum);`

Member Of `XPCAPICOMLib.xPCFileSystem`

Arguments

`[in] scNum`	Enter the scope number.

Return This method 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 `xPCFileSystem.ScGetWriteMode` method returns the write mode of the file for the scope.

See Also API method `xPCFileSystem.ScSetWriteMode`

Purpose

Get block write size of data chunks

Prototype

`long ScGetWriteSize(long *scNum*);`

Member Of

`XPCAPICOMLib.xPCFileSystem`

Arguments

`[in]` *`scNum`*	Enter the scope number.

Return

This method returns the block size, in bytes, of the data chunks.

Description

The `xPCFileSystem.ScGetWriteSize` method gets the block size, in bytes, of the data chunks.

See Also

API method `xPCFileSystem.ScSetWriteSize`

xPCFileSystem.ScSetFileName

Purpose

Specify file name to contain signal data

Prototype

`long ScSetFileName(long scNum, BSTR filename);`

Member Of

`XPCAPICOMLib.xPCFileSystem`

Arguments

`[in]` *`scNum`*	Enter the scope number.
`[in]` *`filename`*	Enter the name of a file to contain the signal data.

Return

If the method detects an error, it returns `-1`. Otherwise, the method returns `0`.

Description

The `xPCFileSystem.ScSetFileName` method 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 method when the scope is stopped.

See Also

API method `xPCFileSystem.ScGetFileName`

Purpose

Specify when file allocation table entry is updated

Prototype

`long ScSetWriteMode(long scNum, long writeMode);`

Member Of

`XPCAPICOMLib.xPCFileSystem`

Arguments

`[in] scNum`	Enter the scope number.	
`[in] writeMode`	Enter an integer for the write mode:	
	`0`	Enables lazy write mode
	`1`	Enables commit write mode

Return

If the method detects an error, it returns `-1`. Otherwise, the method returns `0`.

Description

The `xPCFileSystem.ScSetWriteMode` method 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 method `xPCFileSystem.ScSetWriteMode`

Scope object property `Mode`

xPCFileSystem.ScSetWriteSize

Purpose Specify that memory buffer collect data in multiples of write size

Prototype `long ScSetWriteSize(long scNum, long writeSize);`

Member Of `XPCAPICOMLib.xPCFileSystem`

Arguments

`[in]` *`scNum`*	Enter the scope number.
`[in]` *`writeSize`*	Enter the block size, in bytes, of the data chunks.

Return If the method detects an error, it returns `-1`. Otherwise, the method returns `0`.

Description The `xPCFileSystem.ScSetWriteSize` method 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 method `xPCFileSystem.ScGetWriteSize`

Scope object property `WriteSize`

Purpose

Write to file on target computer

Prototype

```
long WriteFile(long fileHandle, long numbytes,
VARIANT buffer);
```

Member Of

`XPCAPICOMLib.xPCFileSystem`

Arguments

`[in]` *`fileHandle`*	Enter the file handle of an open file on the target computer.
`[in]` *`numbytes`*	Enter the number of bytes this method is to write into the file.
`[in]` *`buffer`*	The contents to write to *`fileHandle`* are stored in *`buffer`*.

Return

If the method detects an error, it returns `-1`. Otherwise, the method returns `0`.

Description

The `xPCFileSystem.WriteFile` method writes the contents of the `VARIANT` *`buffer`*, of type `Byte`, 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 methods `xPCFileSystem.CloseFile`, `xPCFileSystem.GetFileSize`, `xPCFileSystem.OpenFile`, `xPCFileSystem.ReadFile`

xPCProtocol.Close

Purpose Close RS-232 or TCP/IP communication connection

Prototype

```
long Close();
```

Member Of `XPCAPICOMLib.xPCProtocol`

Return If the method detects an error, it returns `0`. Otherwise, it returns `-1`.

Description The `xPCProtocol.Close` method closes the communication channel opened by `xPCProtocol.RS232Connect` or `xPCProtocol.TcpIpConnect`.

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

Purpose

Return current timeout value for target application initialization

Prototype

```
long GetLoadTimeOut();
```

Member Of

`XPCAPICOMLib.xPCProtocol`

Return

If the method detects an error, it returns `-1`. Otherwise, it returns the number of seconds allowed for the initialization of the target application.

Description

The `xPCProtocol.GetLoadTimeOut` method returns the number of seconds allowed for the initialization of the target application.

When you load a new target application onto the target computer, the method `xPCTarget.LoadApp` waits for a certain amount of time before checking to see whether the initialization of the target application is complete. In the case where initialization of the target application is not complete, the method `xPCTarget.LoadApp` returns a timeout error. By default, `xPCTarget.LoadApp` checks five times to see whether the target application is ready, with each attempt taking about 1 second. 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. The method `xPCProtocol.SetLoadTimeOutxPCProtocol.SetLoadTimeOut` sets the timeout to a different number.

Use the `xPCProtocol.GetLoadTimeOut` method if you suspect that the current number of seconds (the timeout value) is too short. Then use the `xxPCProtocol.SetLoadTimeOut` method to set the timeout to a higher number.

xPCProtocol.GetxPCErrorMsg

Purpose Return error string

Prototype `BSTR GetxPCErrorMsg();`

Member Of `XPCAPICOMLib.xPCProtocol`

Return If the `xPCProtocol.GetxPCErrorMsg` method completes without detecting an error, it returns the string for the last reported error.

Description The `xPCProtocol.GetxPCErrorMsg` method returns the string of the last error reported by another COM API method. This value is reset every time you call a new method. Therefore, you should check this constant value immediately after a call to an API COM method. You can use this method in conjunction with the `xPCProtocol.isxPCError` method, which detects that an error has occurred.

See Also API function `xPCProtocol.isxPCError`

Purpose Initialize Simulink Real-Time API DLL

Prototype

```
long Init();
```

Member Of `XPCAPICOMLib.xPCProtocol`

Return If the Simulink Real-Time DLL, `xpcapi.dll` loads without causing `xPCProtocol.Init` to detect an error, the method returns `0`. If `xpcapi.dll` fails to load, this method returns `-1`.

Description The `xPCProtocol.Init` method initializes the Simulink Real-Time API by loading the Simulink Real-Time DLL, `xpcapi.dll`, into memory. To load `xpcapi.dll` into memory, the method requires that the `xpcapi.dll` file be in one of the following folders:

- The folder in which the application is loaded
- The current folder
- The Windows system folder

xPCProtocol.isxPCError

Purpose Return error status

Prototype `long isxPCError();`

Member Of `XPCAPICOMLIB.xPCProtocol`

Return If an error occurred, the method returns `1`. Otherwise, it returns `0`.

Description Use the `xPCProtocol.isxPCError` method to check for errors that might occur after a call to the `xPCProtocol` class methods. If the method detects that an error occurred, call the `xPCProtocol.GetxPCErrorMsg` to get the string for the error.

See Also API function `xPCProtocol.GetxPCErrorMsg`

Purpose Contain communication channel index

Prototype

```
long Port();
```

Member Of

```
XPCAPICOMLIB.xPCProtocol
```

Return If the method detects an error, it returns a nonpositive number. Otherwise, it returns a positive number (the communication channel index).

Description The `xPCProtocol.Port` property contains the communication channel index if connection with the target computer succeeds. Note that you only need to use this property when working with a model-specific COM library that you generate from a Simulink model.

xPCProtocol.Reboot

Purpose Reboot target computer

Prototype `long Reboot();`

Member Of `XPCAPICOMLib.xPCProtocol`

Return If the method detects an error, it returns `0`. Otherwise, it returns `-1`.

Description The `xPCProtocol.Reboot` method reboots the target computer. This function does not close the connection to the target computer. You should explicitly close the connection, then reestablish the connection once the target computer has rebooted. Use the methods `xPCProtocol.RS232Connect` or `xPCProtocol.TcpIpConnect` to reestablish the connection.

Purpose

Open RS-232 connection to target computer

Prototype

```
long RS232Connect(long comport, long baudrate);
```

Member Of

`XPCAPICOMLib.xPCProtocol`

Arguments

[in] *`comport`*	Index of the COM port to be used (`0` is COM1, `1` is COM2, and so forth).
[in] *`baudrate`*	*`baudrate`* must be one of the following values: 1200, 2400, 4800, 9600, 19200, 38400, 57600, or 115200.

Return

The `xPCProtocol.RS232Connect` method returns the port value for the connection. If the method detects an error, it returns `0`. Otherwise, it returns `-1`.

Description

The `xPCProtocol.RS232Connect` method initiates an RS-232 connection to an Simulink Real-Time system. It returns the port value for the connection. Be sure to pass this value to every Simulink Real-Time API function that requires 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 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.

xPCProtocol.SetLoadTimeOut

Purpose

Change initialization timeout value

Prototype

`long SetLoadTimeOut(long` *`timeOut`*`);`

Member Of

`XPCAPICOMLib.xPCProtocol`

Arguments

[in] *`timeOut`*	Enter the new initialization timeout value.

Return

If the method detects an error, it returns `0`. Otherwise, it returns `-1`. To get the string description for the error, use `xPCProtocol.GetxPCErrorMsg`.

Description

The `xPCProtocol.SetLoadTimeOut` method changes the timeout value for initialization. The *`timeOut`* value is the time the method `xPCTarget.LoadApp` waits to check whether the model initialization for a new application is complete before returning. It enables you to set the number of initialization attempts to be made before signaling a timeout. When a new target application is loaded onto the target computer, the method `xPCTarget.LoadApp` 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, `xPCTarget.LoadApp` returns a timeout error.

By default, `xPCTarget.LoadApp` 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.

Purpose

Ping target computer

Prototype

```
long TargetPing;
```

Member Of

`XPCAPICOMLIB.xPCProtocol`

Return

The `xPCProtocol.TargetPing` method does not return an error status. This method returns `1` if it reaches the target computer and the computer responds. If the target computer does not respond, the method returns `0`.

Description

The `xPCProtocol.TargetPing` method pings the target computer and returns `1` or `0` depending on whether the target responds or not. Errors such as the inability to connect to the target are ignored.

If you are using TCP/IP, note that `xPCProtocol.TargetPing` will cause the target computer to close the TCP/IP connection. You can use `xPCProtocol.TcpIpConnect` to reconnect. You can also use this `xPCProtocol.TargetPing` feature to close the target computer connection in the event of an aborted TCP/IP connection (for example, if your host side program crashes).

xPCProtocol.TcpIpConnect

Purpose

Open TCP/IP connection to target computer

Prototype

`long TcpIpConnect(BSTR` *`TargetIpAddress`*`, BSTR` *`TargetPort`*`);`

Member Of

`XPCAPICOMLIB.xPCProtocol`

Arguments

`[in]` *`TargetIpAddress`*	Enter the IP address of the target as a dotted decimal string. For example, `"192.168.0.10"`.
`[in]` *`TargetPort`*	Enter the associated IP port as a string. For example, `"22222"`.

Return

If the method detects an error, it returns `0`. Otherwise, it returns `-1`.

Description

The `xPCProtocol.TcpIpConnect` method opens a connection to the TCP/IP location specified by the IP address. Use this integer as the *`TargetPort`* variable in the Simulink Real-Time COM API functions that require a port value.

Purpose

Unload Simulink Real-Time API DLL from memory

Prototype

```
long Term();
```

Member Of

`XPCAPICOMLib.xPCProtocol`

Return

The `xPCProtocol.Term` method always returns `-1`.

Description

The `xPCProtocol.Term` method unloads the Simulink Real-Time API DLL (`xpcapi.dll`) from memory. You must call this method when you want to terminate your COM API application.

xPCScopes.AddFileScope

Purpose Create new file scope

Prototype `long AddFileScope(long` *`scNum`*`);`

Member Of `XPCAPICOMLib.xPCScopes`

Arguments

[in] *`scNum`*	Enter a number for a new scope. Values are `1`, `2`, `3`. . .

Return If the method detects an error, it returns `0`. Otherwise, it returns `-1`.

Description The `xPCScopes.AddFileScope` method creates a new file scope on the target computer.

Calling the `xPCScopes.AddFileScope` method with *`scNum`* having the number of an existing scope produces an error. Use `xPCScopes.GetScopes` to find the numbers of existing scopes.

Purpose

Create new host scope

Prototype

`long AddHostScope(long *scNum*);`

Member Of

`XPCAPICOMLib.xPCScopes`

Arguments

[in] *`scNum`*	Enter a number for a new scope. Values are `1`, `2`, `3`. . .

Return

If the method detects an error, it returns `0`. Otherwise, it returns `-1`.

Description

The `xPCScopes.AddHostScope` method creates a new host scope on the target computer.

Calling the `xPCScopes.AddHostScope` method with *`scNum`* having the number of an existing scope produces an error. Use `xPCScopes.GetScopes` to find the numbers of existing scopes.

xPCScopes.AddTargetScope

Purpose Create new target scope

Prototype `long AddTargetScope(long scNum);`

Member Of `XPCAPICOMLib.xPCScopes`

Arguments

[in] *scNum*	Enter a number for a new scope. Values are `1`, `2`, `3`. . .

Return If the method detects an error, it returns `0`. Otherwise, it returns `-1`.

Description If the method detects an error, it returns `0`. The `xPCScopes.AddTargetScope` method creates a new scope on the target computer.

Calling the `xPCScopes.AddTargetScope` method with *`scNum`* having the number of an existing scope produces an error. Use `xPCScopes.GetScopes` to find the numbers of existing scopes.

Purpose

Get and copy list of scope numbers

Prototype

`VARIANT GetScopes(long *size*);`

Member Of

`XPCAPICOMLib.xPCScopes`

Arguments

[in] *size*	Specify the size of the `VARIANT` array returned. This argument must be greater than `MAX_SCOPES`-1. The elements in the array consist of a list of unsorted integers, terminated by `-1`.

Return

The `xPCScopes.GetScopes` method returns a `VARIANT` array with elements containing a list of scope numbers from the target application.

Description

The `xPCScopes.GetScopes` method gets a `VARIANT` array with elements containing a list of scope numbers currently defined for the target application. Specify the size of the `VARIANT` array returned. This size must be greater than the maximum number of scopes `-1`, up to a maximum of 30 scopes. The elements in the array consist of a list of unsorted integers, terminated by `-1`.

Purpose Get error string

Prototype `BSTR GetxPCError();`

Member Of `XPCAPICOMLib.xPCScopes`

Return The `xPCScopes.GetxPCError` method returns the string for the last reported error. If the software has not reported an error, this method returns 0.

Description The `xPCScopes.GetxPCError` method gets the string of the last reported error by another COM API method. This value is reset every time you call a new method. Therefore, you should check this constant value immediately after a call to an API COM method. You can use this method in conjunction with the `xPCScopes.isxPCError` method, which detects that an error has occurred.

See Also API function `xPCScopes.isxPCError`

Purpose

Initialize scope object to communicate with target computer

Prototype

```
long Init(IxPCProtocol* xPCProtocol);
```

Member Of

`XPCAPICOMLib.xPCScopes`

Arguments

[in] `xPCProtocol`	Specify the communication port of the target computer object for which the scope is to be initialized.

Return

If the `xPCScopes.Init` method initializes the scope object without detecting an error, it returns `0`. If the scope object fails to initialize, the method returns `-1`.

Description

The `xPCScopes.Init` method initializes the scope object to communicate with the target computer referenced by the `xPCProtocol` object.

xPCScopes.IsScopeFinished

Purpose Get data acquisition status for scope

Prototype `long IsScopeFinished(long scNum);`

Member Of `XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.

Return If the method detects an error, it returns `-1`. If a scope finishes a data acquisition cycle, this method returns `1`. If the scope is in the process of acquiring data, this method returns `0`.

Description The `xPCScopeos.IsScopeFinished` method gets a `1` or `0` 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.

Purpose

Get error status

Prototype

```
long isxPCError();
```

Member Of

`XPCAPICOMLIB.xPCScopes`

Return

If an error occurred, the method returns `1`. Otherwise, it returns `0`.

Description

Use the `xPCScopes.isxPCError` method to check for errors that might occur after a call to the `xPCScopes` class methods. If the software detects that an error occurred, call the `xPCScopes.GetxPCError` method to get the string for the error.

See Also

API function `xPCScopes.GetxPCError`

xPCScopes.RemScope

Purpose Remove scope

Prototype `long RemScope(long scNum);`

Member Of `XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.

Return If the method detects an error, it returns `0`. Otherwise, it returns `-1`.

Description The `xPCScopes.RemScope` method removes the scope with number *scNum*. Attempting to remove a nonexistent scope causes an error. For a list of existing scopes, use `xPCScopes.GetScopes`.

Purpose Add signal to scope

Prototype `long ScopeAddSignal(long scNum, long sigNum);`

Member Of `XPCAPICOMLib.xPCScopes`

Arguments

[in] *`scNum`*	Enter the scope number.
[in] *`sigNum`*	Enter a signal number.

Return If the method detects an error, it returns `0`. Otherwise, it returns `-1`.

Description The `xPCScopes.ScopeAddSignal` method adds the signal with number *`sigNum`* to the scope *`scNum`*. The signal should not already exist in the scope. You can use `xPCScopes.ScopeGetSignals` to get a list of the signals already present. Use the `xPCTarget.GetSignalIdx` method to get the signal number.

xPCScopes.ScopeGetAutoRestart

Purpose Scope autorestart value

Prototype `long ScopeGetAutoRestart(long *scNum*);`

Member Of `XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.

Return The `xPCScopes.ScopeGetAutoRestart` method returns the scope autorestart flag value (`1` if enabled, `0` if disabled). If the method detects an error, it returns `-1`.

Description The `xPCScopes.ScopeGetAutoRestart` method gets the autorestart flag value for scope *scNum*. Autorestart flag can be disabled (`0`) or enabled (`1`).

Purpose

Copy scope data to array

Prototype

```
VARIANT ScopeGetData(long scNum, long signal_id,
long start,
long numsamples, long decimation);
```

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.
[in] *signal_id*	Enter a signal number. Enter `-1` to get time stamped data.
[in] *start*	Enter the first sample from which data retrieval is to start.
[in] *numsamples*	Enter the number of samples retrieved with a decimation of *decimation*, starting from the *start* value.
[in] *decimation*	Enter a value such that every *decimation* sample is retrieved in a scope window.

Return

The `xPCScopes.ScopeGetData` method returns a `VARIANT` array with elements containing the data used in a scope.

Description

The `xPCScopes.ScopeGetData` method 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 `xPCScopes.ScopeGetState` method to check the state of the scope.) The data must be retrieved one signal at a time. The calling function determines and allocates the space ahead of time to store the scope data. Use the method `xPCScopes.ScopeGetSignals` to get the list of signals in the scope for *signal_id*.

To get time stamped data, specify `-1` for `signal_id`. From the output, you can then get the number of nonzero elements.

Purpose

Get decimation of scope

Prototype

`long ScopeGetDecimation(long scNum);`

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.

Return

The `xPCScopes.ScopeGetDecimation` method returns the decimation of scope *scNum*. If the method detects an error, it returns `-1`.

Description

The `xPCScopes.ScopeGetDecimation` method gets the decimation of scope *scNum*. The decimation is a number, `N`, meaning every `N`th sample is acquired in a scope window.

xPCScopes.ScopeGetNumPrePostSamples

Purpose

Get number of pre- or posttriggering samples before triggering scope

Prototype

`long ScopeGetNumPrePostSamples(long *scNum*);`

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.

Return

The `xPCScopes.ScopeGetNumPrePostSamples` method returns the number of samples for pre- or posttriggering for scope *scNum*. If an error occurs, this method returns `-1`.

Description

The `xPCScopes.ScopeGetNumPrePostSamples` method gets the number of samples for pre- or posttriggering for scope *scNum*. A negative number implies pretriggering, whereas a positive number implies posttriggering samples.

Purpose

Get number of samples in one data acquisition cycle

Prototype

```
long ScopeGetNumSamples(long scNum);
```

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

[in] *`scNum`*	Enter the scope number.

Return

The `xPCScopes.ScopeGetNumSamples` method returns the number of samples in the scope *`scNum`*. If the method detects an error, it returns `-1`.

Description

The `xPCScopes.ScopeGetNumSamples` method gets the number of samples in one data acquisition cycle for scope *`scNum`*.

xPCScopes.ScopeGetSignals

Purpose Get list of signals

Prototype `VARIANT ScopeGetSignals(long` `*scNum*`, `long` `*size*`);

Member Of `XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.
[in] *size*	Enter an integer to allocate the number of elements to be returned in the `VARIANT` array. This size is required for the method to copy the list of signals into the `VARIANT` array. The maximum number of signals is 10.

Return The `xPCScopes.ScopeGetSignals` method returns a `VARIANT` array with elements consisting of the list of signals defined for a scope.

Description The `xPCScopes.ScopeGetSignals` method gets the list of signals defined for scope *scNum*. You can use the constant `MAX_SIGNALS`.

xPCScopes.ScopeGetStartTime

Purpose Get last data acquisition cycle start time

Prototype `double ScopeGetStartTime(long scNum);`

Member Of `XPCAPICOMLIB.xPCScopes`

Arguments

[in] *`scNum`*	Enter the scope number.

Return The `xPCScopes.ScopeGetStartTime` method returns the start time for the last data acquisition cycle of a scope. If the method detects an error, it returns `-1`.

Description The `xPCScopes.ScopeGetStartTime` method gets the time at which the last data acquisition cycle for scope *`scNum`* started. This is only valid for scopes of type `SCTYPE_HOST`.

xPCScopes.ScopeGetState

Purpose Get state of scope

Prototype `BSTR ScopeGetState(long *scNum*);`

Member Of `XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.

Return The `xPCScopes.ScopeGetState` method returns the state of scope *scNum*. If the method detects an error, it returns `-1`.

Description The `xPCScopes.ScopeGetState` method gets the state of scope *scNum*, or `-1` upon error.

Constants to find the scope state 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.
`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.

Constant	Value	Description
`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.

xPCScopes.ScopeGetTriggerLevel

Purpose

Get trigger level for scope

Prototype

`double ScopeGetTriggerLevel(long scNum);`

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

[in] *`scNum`*	Enter the scope number.

Return

The `xPCScopes.ScopeGetTriggerLevel` method returns the scope trigger level. If the method detects an error, it returns `-1`.

Description

The `xPCScopes.ScopeGetTriggerLevel` method gets the trigger level for scope *`scNum`*.

Purpose

Get trigger mode for scope

Prototype

`long ScopeGetTriggerMode(long scNum);`

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.

Return

The `xPCScopes.ScopeGetTriggerMode` method returns the scope trigger mode. If the method detects an error, it returns `-1`.

Description

The `xPCScopes.ScopeGetTriggerMode` method gets the trigger mode for scope *scNum*. Use the constants here 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.
`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 function `xPCScopes.ScopeGetTriggerModeStr`

Purpose

Get trigger mode as string

Prototype

`BSTR ScopeGetTriggerModeStr(long scNum);`

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.

Return

The `xPCScopes.ScopeGetTriggerModeStr` method returns a string containing the trigger mode string.

Description

The `xPCScopes.ScopeGetTriggerModeStr` method gets the trigger mode string for scope *scNum*. This method returns one of the following strings.

Constant	Description
`FreeRun`	There is no trigger mode. The scope triggers when it is ready to trigger, regardless of the circumstances.
`Software`	Only user intervention can trigger the scope. No other triggering is possible.
`Signal`	The scope is triggered only after a signal has crossed a value.
`Scope`	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 function `xPCScopes.ScopeGetTriggerMode`

xPCScopes.ScopeGetTriggerSample

Purpose Get sample number for triggering scope

Prototype `long ScopeGetTriggerSample(long scNum);`

Member Of `XPCAPICOMLIB.xPCScopes`

Arguments

[in] *`scNum`*	Enter the scope number.

Return The `xPCScopes.ScopeGetTriggerSample` method 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 method detects an error, it returns `-1`.

Description The `xPCScopes.ScopeGetTriggerSample` method 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.

Purpose

Get trigger signal for scope

Prototype

`long ScopeGetTriggerSignal(long scNum);`

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.

Return

The `xPCScopes.ScopeGetTriggerSignal` method returns the scope trigger signal. If the method detects an error, it returns `-1`.

Description

The `xPCScopes.ScopeGetTriggerSignal` method gets the trigger signal for scope *scNum*.

xPCScopes.ScopeGetTriggerSlope

Purpose Get trigger slope for scope

Prototype `long ScopeGetTriggerSlope(long scNum);`

Member Of `XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.

Return The `xPCScopes.ScopeGetTriggerSlope` method returns the scope trigger slope. If the method detects an error, it returns `-1`.

Description The `xPCScopes.ScopeGetTriggerSlope` method gets the trigger slope of scope *scNum*. Use the constants here to interpret the trigger slope:

String	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 function `xPCScopes.ScopeGetTriggerSlopeStr`

Purpose

Get trigger slope as string

Prototype

`BSTR ScopeGetTriggerSlopeStr(long *scNum*);`

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.

Return

The `xPCScopes.ScopeGetTriggerSlopeStr` method returns a string containing the trigger slope string.

Description

The `xPCScopes.ScopeGetTriggerSlopeStr` method gets the trigger slope string for scope *scNum*. This method returns one of the following strings:

String	Description
`Either`	The trigger slope can be either rising or falling.
`Rising`	The trigger slope must be rising when the signal crosses the trigger value.
`Falling`	The trigger slope must be falling when the signal crosses the trigger value.

See Also

API function `xPCScopes.ScopeGetTriggerSlope`

xPCScopes.ScopeGetType

Purpose Get type of scope

Prototype `BSTR ScopeGetType(long *scNum*);`

Member Of `XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.

Return The `xPCScopes.ScopeGetType` method returns the scope type as a string. If the method detects an error, it returns `-1`.

Description The `xPCScopes.ScopeGetType` method gets the type of scope *scNum*. This method returns one of the following strings:

String	Description
`HOST`	Host scope
`Target`	Target scope

xPCScopes.ScopeRemSignal

Purpose Remove signal from scope

Prototype `long ScopeRemSignal(long *scNum*, long *sigNum*);`

Member Of `XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.
[in] *sigNum*	Enter a signal number.

Return If the method detects an error, it returns `0`. Otherwise, it returns `-1`.

Description The `xPCScopes.ScopeRemSignal` method removes a signal from the scope with number *scNum*. The scope must already exist, and signal number *sigNum* must exist in the scope. Use `xPCScopes.GetScopes` to determine the existing scopes, and use `xPCScopes.ScopeGetSignals` to determine the existing signals for a scope. Use this function only when the scope is stopped. Use `xPCScopes.ScopeGetState` to check the state of the scope.

xPCScopes.ScopeSetAutoRestart

Purpose Scope autorestart value

Prototype `long ScopeSetAutoRestart(long scNum, long onoff);`

Member Of `XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.
[in] *onoff*	Enter value to enable (1) or disable (0) scope autorestart.

Return The `xPCScopes.ScopeSetAutoRestart` method returns the scope autorestart flag value (`1` if enabled, `0` if disabled). If the method detects an error, it returns `-1`.

Description The `xPCScopes.ScopeSetAutoRestart` method sets the autorestart flag value for scope *scNum*. Autorestart flag can be disabled (`0`) or enabled (`1`).

Purpose

Set decimation of scope

Prototype

```
long ScopeSetDecimation(long scNum, long decimation);
```

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

[in] *`scNum`*	Enter the scope number.
[in] *`decimation`*	Enter an integer for the decimation.

Return

If the method detects an error, it returns `0`. Otherwise, it returns `-1`.

Description

The `xPCScopes.ScopeSetDecimation` method 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 `xPCScopes.ScopeGetState` to check the state of the scope.

xPCScopes.ScopeSetNumPrePostSamples

Purpose Set number of pre- or posttriggering samples before triggering scope

Prototype `long ScopeSetNumPrePostSamples(long scNum, long prepost);`

Member Of `XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.
[in] *prepost*	A negative number means pretriggering, while a positive number means posttriggering. This function can only be used when the scope is stopped.

Return If the method detects an error, it returns `0`. Otherwise, it returns `-1`.

Description The `xPCScopes.ScopeSetNumPrePostSamples` method sets the number of samples for pre- or posttriggering for scope *scNum* to *prepost*. Use this method only when the scope is stopped. Use `xPCScopes.ScopeGetState` to check the state of the scope. Use the `xPCScopes.GetScopes` method to get a list of scope numbers.

xPCScopes.ScopeSetNumSamples

Purpose Set number of samples in one data acquisition cycle

Prototype `long ScopeSetNumSamples(long scNum, long samples);`

Member Of `XPCAPICOMLIB.xPCScopes`

Arguments

[in] *`scNum`*	Enter the scope number.
[in] *`samples`*	Enter the number of samples you want to acquire in one cycle.

Return If the method detects an error, it returns `0`. Otherwise, it returns `-1`.

Description The `xPCScopes.ScopeSetNumSamples` method sets the number of samples for scope *`scNum`* to *`samples`*. Use this function only when the scope is stopped. Use `xPCScopes.ScopeGetState` to check the state of the scope.

xPCScopes.ScopeSetTriggerLevel

Purpose Set trigger level for scope

Prototype `long ScopeSetTriggerLevel(long scNum, double level);`

Member Of `XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.
[in] *level*	Value for a signal to trigger data acquisition with a scope.

Return If the method detects an error, it returns `0`. Otherwise, it returns `-1`.

Description The `xPCScopes.ScopeSetTriggerLevel` method sets the trigger level to *level* for scope *scNum*. Use this function only when the scope is stopped. Use `xPCScopes.ScopeGetState`to check the state of the scope.

Purpose

Set trigger mode of scope

Prototype

`long ScopeSetTriggerMode(long *scNum*, long *triggermode*);`

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.
[in] *triggermode*	Trigger mode for a scope.

Return

If the method detects an error, it returns `0`. Otherwise, it returns `-1`.

Description

The `xPCScopes.ScopeSetTriggerMode` method sets the trigger mode of scope *scNum* to *triggermode*. Use this method only when the scope is stopped. Use `xPCScopes.ScopeGetState`to check the state of the scope. Use the `xPCScopes.GetScopes` method to get a list of scopes.

Use the constants defined here 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.

Constant	Value	Description
`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`).

xPCScopes.ScopeSetTriggerSample

Purpose

Set sample number for triggering scope

Prototype

`long ScopeSetTriggerSample(long scNum, long trigScSample);`

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.
[in] *trigScSample*	Enter a nonnegative integer for the number of samples acquired by the triggering scope before starting data acquisition on a second scope.

Return

If the method detects an error, it returns `0`. Otherwise, it returns `-1`.

Description

The `xPCScopes.ScopeSetTriggerSample` method sets the number of samples (*trigScSample*) a triggering scope acquires before it triggers a second scope (*scNum*). Use the `xPCScopes.GetScopes` method to get a list of scopes.

For meaningful results, set *trigScSample* 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, use a value of `-1` for *trigScSamp*.

xPCScopes.ScopeSetTriggerSignal

Purpose

Select signal to trigger scope

Prototype

`long ScopeSetTriggerSignal(long *scNum*, long *triggerSignal*);`

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

[in] *`scNum`*	Enter the scope number.
[in] *`trigSignal`*	Enter a signal number.

Return

If the method detects an error, it returns `0`. Otherwise, it returns `-1`.

Description

The `xPCScopes.ScopeSetTriggerSignal` method sets the trigger signal of scope *`scNum`* to *`trigSig`*. The trigger signal *`trigSig`* must be one of the signals in the scope. Use this method only when the scope is stopped. You can use `xPCScopes.ScopeGetSignals` to get the list of signals in the scope. Use`xPCScopes.ScopeGetState` to check the state of the scope. Use the `xPCScopes.GetScopes` method to get a list of scopes.

Purpose

Set slope of signal that triggers scope

Prototype

`long ScopeSetTriggerSlope(long scNum, long triggerslope);`

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

[in] *`scNum`*	Enter the scope number.
[in] *`triggerSlope`*	Enter the slope mode for the signal that triggers the scope.

Return

If the method detects an error, it returns `0`. Otherwise, it returns `-1`.

Description

The `xPCScopes.ScopeSetTriggerSlope` method sets the trigger slope of scope *`scNum`* to *`trigSlope`*. Use this method only when the scope is stopped. Use `xPCScopes.ScopeGetState` to check the state of the scope. Use the `xPCScopes.GetScopes` method to get a list of scopes.

Use the constants defined here 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.

xPCScopes.ScopeSoftwareTrigger

Purpose Set software trigger of scope

Prototype `long ScopeSoftwareTrigger(long scNum);`

Member Of `XPCAPICOMLIB.xPCScopes`

Arguments

[in] *`scNum`*	Enter the scope number.

Return If the method detects an error, it returns `0`. Otherwise, it returns `-1`.

Description The `xPCScopes.ScopeSoftwareTrigger` method triggers scope *`scNum`*. The scope must be in the state `Waiting for trigger` for this method to succeed. Use `xPCScopes.ScopeGetState` to check the state of the scope. Use the `xPCScopes.GetScopes` method to get a list of scopes.

You can use the `xPCScopes.ScopeSoftwareTrigger` method to trigger the scope, regardless of the trigger mode.

Purpose

Start data acquisition for scope

Prototype

`long ScopeStart(long scNum);`

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.

Return

If the method detects an error, it returns `0`. Otherwise, it returns `-1`.

Description

The `xPCScopes.ScopeStart` method 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 `xPCScopes.ScopeGetState` to check the state of the scope or, for host scopes that are already started, call `xPCScopes.IsScopeFinished`. Use the `xPCScopes.GetScopes` method to get a list of scopes.

xPCScopes.ScopeStop

Purpose Stop data acquisition for scope

Prototype `long ScopeStop(long scNum);`

Member Of `XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.

Return If the method detects an error, it returns `0`. Otherwise, it returns `-1`.

Description The `xPCScopes.ScopeStop` method stops the scope *scNum*. This sets the scope to the `Interrupted` state. The scope must be running for this function to succeed. Use `xPCScopes.ScopeGetState` to determine the state of the scope. Use the `xPCScopes.GetScopes` method to get a list of scopes.

Purpose

Get status of grid line for particular scope

Prototype

`long TargetScopeGetGrid(long scNum);`

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.

Return

The `xPCScopes.TargetScopeGetGrid` method returns the state of the grid lines for scope *scNum*. If the method detects an error, it returns `-1`.

Description

The `xPCScopes.TargetScopeGetGrid` method 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 the `xPCScopes.GetScopes` method to get a list of scopes.
- Use `xPCScopes.TargetScopeGetMode` and `xPCScopes.TargetScopeSetMode` to retrieve and set the scope mode.

xPCScopes.TargetScopeGetMode

Purpose Get scope mode for displaying signals

Prototype `long TargetScopeGetMode(long *scNum*);`

Member Of `XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.

Return The `xPCScopes.TargetScopeGetMode` method 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 the method detects an error, it returns `-1`.

Description The `xPCScopes.TargetScopeGetMode` method gets the mode of the scope *scNum*, which must be of type `SCTYPE_TARGET`. Use the `xPCScopes.GetScopes` method to get a list of scopes.

See Also API function `xPCScopes.TargetScopeGetModeStr`

xPCScopes.TargetScopeGetModeStr

Purpose

Get scope mode string for displaying signals

Prototype

`BSTR TargetScopeGetModeStr(long` *`scNum`*`);`

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

[in] *`scNum`*	Enter the scope number.

Return

The `xPCScopes.TargetScopeGetModeStr` method returns the string corresponding to the scope mode. The possible strings are

- `Numerical`
- `Redraw`
- `Sliding`
- `Rolling`

Description

The `xPCScopes.TargetScopeGetModeStr` method gets the mode string of the scope *`scNum`*, which must be of type `SCTYPE_TARGET`. Use the `xPCScopes.GetScopes` method to get a list of scopes.

See Also

API function `xPCScopes.TargetScopeGetMode`

xPCScopes.TargetScopeGetViewMode

Purpose

Get view mode for target computer display

Prototype

```
long TargetScopeGetViewMode();
```

Member Of

`XPCAPICOMLIB.xPCScopes`

Return

The `xPCScopes.TargetScopeGetViewMode` method returns the view mode for the target computer screen. If the method detects an error, it returns `-1`.

Description

The `xPCScopes.TargetScopeGetViewMode` method gets the view (zoom) mode for the target computer display. If the returned value is not zero, the number is 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).

xPCScopes.TargetScopeGetYLimits

Purpose

Get *y*-axis limits for scope

Prototype

```
VARIANT TargetScopeGetYLimits(long scNum);
```

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

[in] *`scNum`*	Enter the scope number.

Return

The `xPCScopes.TargetScopeGetYLimits` method returns the upper and lower limits for target scopes.

Description

The `xPCScopes.TargetScopeGetYLimits` method gets and copies the upper and lower limits for a scope of type `SCTYPE_TARGET` and with scope number *`scNum`*. If both elements are zero, the limits are autoscaled. Use the `xPCScopes.GetScopes` method to get a list of scopes.

Purpose

Set grid mode for scope

Prototype

`long TargetScopeSetGrid(long scNum, long gridonoff);`

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.
[in] *gridonoff*	Enter a grid value (`0` or `1`).

Return

If the method detects an error, it returns `0`. Otherwise, it returns `-1`.

Description

The `xPCScopes.TargetScopeSetGrid` method sets the grid of a scope of type `SCTYPE_TARGET` and scope number *scNum* to *gridonoff*. If *gridonoff* is `0`, the grid is off. If *gridonoff* 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 `xPCScopes.GetScopes` method to get a list of scopes.

Purpose Set display mode for scope

Prototype `long TargetScopeSetMode(long scNum, long mode);`

Member Of `XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.
in] *mode*	Enter the value for the mode.

Return If the method detects an error, it returns `0`. Otherwise, it returns `-1`.

Description The `xPCScopes.TargetScopeSetMode` method 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 `xPCScopes.GetScopes` method to get a list of scopes.

xPCScopes.TargetScopeSetViewMode

Purpose Set view mode for scope

Prototype

```
long TargetScopeSetViewMode(long scNum);
```

Member Of `XPCAPICOMLIB.xPCScopes`

Arguments

[in] *`scNum`*	Enter the scope number.

Return If the method detects an error, it returns `0`. Otherwise, it returns `-1`.

Description The `xPCScopes.TargetScopeSetViewMode` method 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 `xPCScopes.GetScopes` method to get a list of scopes.

Purpose

Set *y*-axis limits for scope

Prototype

```
long TargetScopeSetYLimits(long scNum, SAFEARRAY(double)*
Ylimitarray);
```

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

[in] *`scNum`*	Enter the scope number.
[in, out] *`Ylimitarray`*	Enter a two-element array.

Return

If the method detects an error, it returns `0`. Otherwise, it returns `-1`.

Description

The `xPCScopes.TargetScopeSetYLimits` method sets the *y*-axis limits for a scope with scope number *`scNum`* and type `SCTYPE_TARGET` to the values in the double array *`YlimitArray`*. 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 `xPCScopes.GetScopes` method to get a list of scopes.

xPCTarget.AverageTET

Purpose

Get average task execution time

Prototype

```
double AverageTET();
```

Member Of

`XPCAPICOMLib.xPCTarget`

Return

The `xPCTarget.AverageTET` method returns the average task execution time (TET) for the target application. If the method detects an error, it returns `-1`.

Description

The `xPCTarget.AverageTET` method gets the TET for the target application. You can use this function when the target application is running or when it is stopped.

Purpose

Get target application name

Prototype

```
BSTR GetAppName();
```

Member Of

`XPCAPICOMLib.xPCTarget`

Return

The `xPCTarget.GetAppName` method returns a string with the name of the target application.

Description

The `xPCTarget.GetAppName` method gets the name of the target application. You can use the return value, ***`model_name`***, in a `printf` or similar statement. In case of error, the string is unchanged. Be sure to allocate enough space to accommodate the longest target name you have.

xPCTarget.GetExecTime

Purpose

Get execution time for target application

Prototype

```
double GetExecTime();
```

Member Of

`XPCAPICOMLib.xPCTarget`

Return

The `xPCTarget.GetExecTime` method returns the current execution time for a target application. If the method detects an error, it returns `-1`.

Description

The `xPCTarget.GetExecTime` method gets the current execution time for the running target application. If the target application is stopped, the value is the last running time when the target application was stopped. If the target application is running, the value is the current running time.

xPCTarget.GetNumOutputs

Purpose

Get number of outputs

Prototype

```
long GetNumOutputs();
```

Member Of

`XPCAPICOMLib.xPCTarget`

Return

The `xPCTarget.GetNumOutputs` method returns the number of outputs in the current target application. If the method detects an error, it returns `-1`.

Description

The `xPCTarget.GetNumOutputs` method gets the number of outputs in the target 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.

xPCTarget.GetNumParams

Purpose Get number of tunable parameters

Prototype `long GetNumParams();`

Member Of `XPCAPICOMLib.xPCTarget`

Return The `xPCTarget.GetNumParams` method returns the number of tunable parameters in the target application. If the method detects an error, it returns `-1`.

Description The `xPCTarget.GetNumParams` method gets the number of tunable parameters in the target application. Use this method to see how many parameters you can get or modify.

Purpose

Get number of signals

Prototype

```
long GetNumSignals();
```

Member Of

`XPCAPICOMLib.xPCTarget`

Return

The `xPCTarget.GetNumSignals` method returns the number of signals in the target application. If the method detects an error, it returns `-1`.

Description

The `xPCTarget.GetNumSignals` method gets the total number of signals in the target application that can be monitored from the host. Use this method to see how many signals you can monitor.

xPCTarget.GetNumStates

Purpose

Get number of states

Prototype

`long GetNumStates();`

Member Of

`XPCAPICOMLib.xPCTarget`

Return

The `xPCTarget.GetNumStates` method returns the number of states in the target application. If the method detects an error, it returns `-1`.

Description

The `xPCTarget.GetNumStates` method gets the number of states in the target application.

Purpose

Copy output log data to array

Prototype

```
VARIANT GetOutputLog(long start, long numsamples,
long decimation,
long output_id);
```

Member Of

`XPCAPICOMLib.xPCTarget`

Arguments

[in] *`start`*	Enter the index of the first sample to copy.
[in] *`numsamples`*	Enter the number of samples to copy from the output log.
[in] *`decimation`*	Select whether to copy all the sample values or every `N`th value.
[in] *`output_id`*	Enter an output identification number.

Return

The `xPCTarget.GetOutputLog` method returns output log data. You get the data for each output signal. If the method detects an error, it returns `VT_ERROR`, a scalar.

Description

The `xPCTarget.GetOutputLog` method gets the output log and copies that log to an array. Output IDs range from `0` to (`N`-1), where `N` is the return value of `xPCTarget.GetNumOutputs`. Entering `1` for *`decimation`* copies all values. Entering `N` copies every `N`th value.

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

Note that the target application must be stopped before you get the output log data.

xPCTarget.GetParam

Purpose

Get parameter values

Prototype

`VARIANT GetParam(long *paramIdx*);`

Member Of

`XPCAPICOMLib.xPCTarget`

Arguments

[in] *`paramIdx`*	Enter the index for a parameter.

Return

The `xPCTarget.GetParam` method returns the parameter values of a parameter.

Description

The `xPCTarget.GetParam` method gets the parameter values of a parameter identified by *`paramIdx`*. This method returns an array of type `VARIANT` containing the parameter values, with the conversion of the values being done in column-major format. Each element in the array is a double, regardless of the data type of the actual parameter. You can query the dimensions of the array by calling the method `xPCTarget.GetParamDims`. See the Microsoft Visual Basic .NET 2003 solution located in *`matlabroot`*`\toolbox\rtw\targets\xpc\api\VBNET\SigsAndParamsDemo` for an example of how to use this method.

See Also

API method `xPCTarget.GetParamDims`, `xPCTarget.SetParam`

Purpose

Get row and column dimensions of parameter

Prototype

`VARIANT GetParamDims(long paramIdx);`

Member Of

`XPCAPICOMLib.xPCTarget`

Arguments

[in] *`paramIdx`*	Parameter index.

Return

The `xPCTarget.GetParamDims` method returns a `VARIANT` array of two elements.

Description

The `xPCTarget.GetParamDims` method gets a `VARIANT` array of two elements. The first element contains the number of rows of the parameter, the second element contains the number of columns for your parameter.

xPCTarget.GetParamIdx

Purpose Get parameter index

Prototype `long GetParamIdx(BSTR blockName, BSTR paramName);`

Member Of `XPCAPICOMLib.xPCTarget`

Arguments

[in] *blockName*	Enter the full block path generated by the Simulink Coder software.
[in] *paramName*	Enter the parameter name for a parameter associated with the block.

Return The `xPCTarget.GetParamIdx` method returns the parameter index for the parameter name. If the method detects an error, it returns `-1`.

Description The `xPCTarget.GetParamIdx` method gets 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 target 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.

Purpose

Get parameter name

Prototype

`VARIANT GetParamName(long` *`paramIdx`*`);`

Member Of

`XPCAPICOMLib.xPCTarget`

Arguments

[in] *`paramIdx`*	Enter a parameter index.

Return

The `xPCTarget.GetParamName` method returns a `VARIANT` array that contains two elements, the block path and parameter name, as strings.

Description

The `xPCTarget.GetParamName` method gets the parameter name and block name for a parameter with the index *`paramIdx`*. If *`paramIdx`* is invalid, `xPCGetLastError` returns nonzero, and the strings are unchanged. Get the parameter index with the method `xPCTarget.GetParamIdx`.

Purpose Get sample time

Prototype

```
double GetSampleTime();
```

Member Of `XPCAPICOMLib.xPCTarget`

Return The `xPCTarget.GetSampleTime` method returns the sample time, in seconds, of the target application. If the method detects an error, it returns `-1`.

Description The `xPCTarget.GetSampleTime` method gets the sample time, in seconds, of the target application. You can get the error by using the method `xPCGetLastError`.

Purpose

Get signal value

Prototype

`double GetSignal(long` *`sigNum`*`);`

Member Of

`XPCAPICOMLib.xPCTarget`

Arguments

[in] *`sigNum`*	Enter a signal number.

Return

The `xPCTarget.GetSignal` method returns the current value of signal *`sigNum`*. If the method detects an error, it returns `-1`.

Description

The `xPCTarget.GetSignal` method gets the current value of a signal. Use the `xPCTarget.GetSignalIdx` method to get the signal number.

xPCTarget.GetSignalidsfromLabel

Purpose

Get signal IDs from signal label

Prototype

`VARIANT GetSignalidsfromLabel(BSTR *sigLabel*);`

Member Of

`XPCAPICOMLib.xPCTarget`

Arguments

[in] *`sigLabel`*	Enter a signal label.

Return

The `xPCTarget.GetSignalidsfromLabel` method returns a VARIANT array of the signal elements contained in the signal *`sigLabel`*. If no labels exist, the method returns an empty string.

Description

The `xPCTarget.GetSignalidsfromLabel` method returns a VARIANT array of the signal elements contained in the signal *`sigLabel`*. Signal labels must be unique.

This method 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.

See Also

API method `xPCTarget.GetSignalLabel`

Purpose

Get signal label

Prototype

`BSTR GetSignalLabel(long *sigIdx*);`

Member Of

`XPCAPICOMLib.xPCTarget`

Arguments

[in] *`sigIdx`*	Enter a signal index.

Return

The `xPCTarget.GetSignalLabel` method returns the label of the signal. If no labels exist, the method returns an empty string.

Description

The `xPCTarget.GetSignalLabel` method copies and gets the signal label of a signal with *`sigIdx`*. The method returns the signal label. This method assumes that you already know the signal index. Signal labels must be unique.

This method 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.

See Also

API method `xPCTarget.GetSignalidsfromLabel`

xPCTarget.GetSignalIdx

Purpose Get signal index

Prototype `long GetSignalIdx(BSTR sigName);`

Member Of `XPCAPICOMLib.xPCTarget`

Arguments

[in] *`sigName`*	Enter a signal name.

Return The `xPCTarget.GetSignalIdx` method returns the index for the signal with name *`sigName`*. If the method detects an error, it returns `-1`.

Description The `xPCTarget.GetSignalIdx` method gets 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.

Purpose

Copy signal name to character array

Prototype

```
BSTR GetSignalName(long sigIdx);
```

Member Of

`XPCAPICOMLib.xPCTarget`

Arguments

[in] *`sigIdx`*	Enter a signal index.

Return

The `xPCTarget.GetSignalName` method returns the name of the signal.

Description

The `xPCTarget.GetSignalName` method copies and gets the signal name, including the block path, of a signal with *`sigIdx`*. The method returns a signal name, which makes it convenient to use in a `printf` or similar statement. This method assumes that you already know the signal index.

xPCTarget.GetSignals

Purpose

Get vector of signal values

Prototype

```
VARIANT GetSignals(long NumOfSignals, SAFEARRAY(int)*
SignalsIdxArray);
```

Member Of

`XPCAPICOMLib.xPCTarget`

Arguments

[in] *`NumOfSignals`*	Enter the number of signals to acquire (the number of IDs in *`SignalsIdxArray`*).
[out] *`SignalsIdxArray`*	Enter the IDs of the signals to acquire.

Return

The `xPCTarget.GetSignals` method returns a double-valued variant array containing the current value of a vector of signals. If the method detects an error, it returns `VT_ERROR`, a scalar.

Description

This function returns the values of a vector of up to 1000 signals as fast as it can acquire them. The values are converted to doubles regardless of the actual data type of the signal.

Tip

- Pass an integer array of signal numbers into *`SignalsIdxArray`*. Get the signal numbers with the function `xPCTarget.GetSignalIdx`.
- The signal values may not be at the same time step. To get signal values at the same time step, define a scope of type `SCTYPE_HOST` and use `xPCScopes.ScopeGetData`.

The function `xPCTarget.GetSignal` does the same thing for a single signal, and could be used multiple times to achieve the same result.

However, xPCGetSignals is faster and the signal values are more likely to be spaced closely together.

See Also API functions `xPCTarget.GetSignal`, `xPCTarget.GetSignalIdx`

xPCTarget.GetSignalWidth

Purpose

Get width of signal

Prototype

`long GetSignalWidth(long *sigIdx*);`

Member Of

`XPCAPICOMLib.xPCTarget`

Arguments

[in] *`sigIdx`*	Enter the index of a signal.

Return

The `xPCTarget.GetSignalWidth` method returns the signal width for a signal with *`sigIdx`*. If the method detects an error, it returns `-1`.

Description

The `xPCTarget.GetSignalWidth` method gets 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. A signal's width is the number of signals in the vector.

Purpose

Get state log

Prototype

```
VARIANT GetStateLog(long start, long numsamples,
long decimation,
long state_id);
```

Member Of

`XPCAPICOMLib.xPCTarget`

Arguments

[in] *start*	Enter the index of the first sample to copy.
[in] *numsamples*	Enter the number of samples to copy from the output log.
[in] *decimation*	Select whether to copy all the sample values or every `N`th value.
[in] *state_id*	Enter a state identification number.
`[out, retval]` *Outarray*	The log is stored in *Outarray*, whose allocation is the responsibility of the caller.

Return

The `xPCTarget.GetStateLog` method returns the state log. If the method detects an error, it returns `VT_ERROR`, a scalar.

Description

The `xPCTarget.GetStateLog` method gets the state log. 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 `xPCTarget.GetNumStates`. Entering `1` for *decimation* copies all values. Entering `N` copies every `N`th value. For *start*, the sample indices range from `0` to (`N`-1), where `N` is the return value of `xPCTarget.NumLogSamples`. Use the `xPCTarget.NumLogSamples` method to get the maximum number of samples.

Note that the target application must be stopped before you get the number.

Purpose

Get stop time

Prototype

```
double GetStopTime();
```

Member Of

`XPCAPICOMLib.xPCTarget`

Return

The `xPCTarget.GetStopTime` method returns the stop time as a double, in seconds, of the target application. If the method detects an error, it returns `-1`.

Description

The `xPCTarget.GetStopTime` method gets the stop time, in seconds, of the target application. This is the amount of time the target application runs before stopping.

Purpose

Get TET log

Prototype

```
VARIANT GetTETLog(long start, long numsamples,
long decimation);
```

Member Of

`XPCAPICOMLib.xPCTarget`

Arguments

[in] *start*	Enter the index of the first sample to copy.
[in] *numsamples*	Enter the number of samples to copy from the TET log.
[in] *decimation*	Select whether to copy all the sample values or every `N`th value.
`[out, retval]` *Outarray*	The log is stored in *Outarray*, whose allocation is the responsibility of the caller.

Return

The `xPCTarget.GetTETLog` method returns the TET log. If the method detects an error, it returns `VT_ERROR`, a scalar.

Description

The `xPCTarget.GetTETLog` method gets the task execution time (TET) log. Entering `1` for *decimation* copies all values. Entering `N` copies every `N`th value. For *start*, the sample indices range from `0` to (`N`-1), where `N` is the return value of `xPCTarget.NumLogSamples`. Use the `xPCTarget.NumLogSamples` method to get the maximum number of samples.

Note that the target application must be stopped before you get the number.

xPCTarget.GetTimeLog

Purpose

Get time log

Prototype

```
VARIANT GetTimeLog(long start, long numsamples,
long decimation);
```

Member Of

`XPCAPICOMLib.xPCTarget`

Arguments

[in] *start*	Enter the index of the first sample to copy.
[in] *numsamples*	Enter the number of samples to copy from the time log.
[in] *decimation*	Select whether to copy all the sample values or every Nth value.

Return

The `xPCTarget.GetTimeLog` method returns the time log. If the method detects an error, it returns `VT_ERROR`, a scalar.

Description

The `xPCTarget.GetTimeLog` method gets the time log. 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 *start*, the sample indices range from `0` to (`N`-1), where `N` is the return value of `xPCTarget.NumLogSamples`. Use the `xPCTarget.NumLogSamples` method to get the number of samples.

Note that the target application must be stopped before you get the number.

Purpose Get error string

Prototype `BSTR GetxPCError();`

Member Of `XPCAPICOMLib.xPCTarget`

Return The `xPCTarget.GetxPCError` method returns the string for the last reported error. If the software has not reported an error, this method returns `0`.

Description The `xPCTarget.GetxPCError` method gets the string of the error last reported by another COM API method. This value is reset every time you call a new method. Therefore, you should check this constant value immediately after a call to an API COM method. You can use this method in conjunction with the `xPCTarget.isxPCError` method, which detects that an error has occurred.

See Also API method `xPCTarget.isxPCError`

xPCTarget.Init

Purpose Initialize target object to communicate with target computer

Prototype

```
long Init(IxPCProtocol* xPCProtocol);
```

Member Of `XPCAPICOMLib.xPCTarget`

Return If the method detects an error, it returns `-1`. Otherwise, it returns `0`.

If the `xPCTarget.Init` method initializes the target object without detecting an error, it returns `0`. If the target object fails to initialize, this method returns `-1`.

Description The `xPCTarget.Init` method initializes the target object to communicate with the target computer referenced by the `xPCProtocol` object.

Purpose Return running status for target application

Prototype

```
long IsAppRunning();
```

Member Of `XPCAPICOMLib.xPCTarget`

Return If the target application is stopped, the `xPCTarget.IsAppRunning` method returns `0`. If the target application is running, this method returns `1`. If the method detects an error, it returns `-1`.

Description The `xPCTarget.IsAppRunning` method returns `1` or `0` depending on whether the target application is stopped or running.

xPCTarget.IsOverloaded

Purpose Return overload status for target computer

Prototype

```
long IsOverloaded();
```

Member Of `XPCAPICOMLib.xPCTarget`

Return If the target application has overloaded the CPU, the `xPCTarget.IsOverloaded` method returns `1`. If it has not overloaded the CPU, the method returns `0`. If the method detects an error, it returns `-1`.

Description The `xPCTarget.IsOverloaded` method checks if the target application has overloaded the target computer and returns `1` if it has and `0` if it has not. If the target application is not running, the method returns `0`.

Purpose Return error status

Prototype
```
long isxPCError();
```

Member Of `XPCAPICOMLIB.xPCTarget`

Return If an error occurred, the method returns `1`. Otherwise, it returns `0`.

Description Use the `xPCTarget.isxPCError` method to check for errors that might occur after a call to the `xPCTarget` class methods. If the method detects that an error occurred, call the `xPCTarget.GetxPCError` method to get the string for the error.

See Also API method `xPCTarget.GetxPCError`

Purpose Load target application onto target computer

Prototype `long LoadApp(BSTR pathstr, BSTR filename);`

Member Of `XPCAPICOMLIB.xPCTarget`

Arguments

[in] *`pathstr`*	Enter the full path to the target application file, excluding the file name. For example, in C, use a string like `"C:\\work"`, in Microsoft Visual Basic, use a string like `'C:\work'`.
[in] *`filename`*	Enter the name of a compiled target application (`*.dlm`) without the file extension. For example, in C use a string like `"xpcosc"`, in Microsoft Visual Basic, use a string like `'xpcosc'`.

Return If the method detects an error, it returns `0`. Otherwise, it returns `-1`.

Description The `xPCTarget.LoadApp` method loads the compiled target 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 target application extension.

Before returning, `xPCTarget.LoadApp` waits for a certain amount of time before checking whether the model initialization is complete. In the case where the model initialization is incomplete, `xPCTarget.LoadApp` returns a timeout error to indicate a connection problem (for example, `ETCPREAD`). By default, `xPCTarget.LoadApp` 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 methods `xPCProtocol.GetLoadTimeOut` and `xPCProtocol.SetLoadTimeOut` control the number of attempts made.

xPCTarget.MaximumTET

Purpose

Copy maximum task execution time to array

Prototype

```
VARIANT MaximumTET();
```

Member Of

`XPCAPICOMLIB.xPCTarget`

Return

The `xPCTarget.MaximumTET` method returns a `VARIANT` object containing the maximum task execution time (TET) and the time at which the maximum TET was achieved. The maximum TET value is copied into the first element, and the time at which it was achieved is copied into the second element.

Description

The `xPCTarget.MaximumTET` method returns the maximum TET that was achieved during the previous target application run.

Purpose

Return maximum number of samples that can be in log buffer

Prototype

```
long MaxLogSamples();
```

Member Of

`XPCAPICOMLIB.xPCTarget`

Return

The `xPCTarget.MaxLogSamples` method returns the total number of samples. If the method detects an error, it returns `-1`.

Description

The `xPCTarget.MaxLogSamples` method returns the total number of samples that can be returned in the logging buffers.

Note that the target application must be stopped before you get the number.

Purpose

Copy minimum task execution time to array

Prototype

```
VARIANT MinimumTET();
```

Member Of

`XPCAPICOMLIB.xPCTarget`

Return

The `xPCTarget.MinimumTET` method returns a `VARIANT` object containing the minimum task execution time (TET) and the time at which the minimum TET was achieved. The minimum TET value is copied into the first element, and the time at which it was achieved is copied into the second element.

Description

The `xPCTarget.MinimumTET` method returns the minimum task execution time (TET) that was achieved during the previous target application run.

Purpose

Return number of samples in log buffer

Prototype

```
long NumLogSamples();
```

Member Of

`XPCAPICOMLIB.xPCTarget`

Return

The `xPCTarget.NumLogSamples` method returns the number of samples in the log buffer. If the method detects an error, it returns `-1`.

Description

The `xPCTarget.NumLogSamples` method returns the number of samples in the log buffer. In contrast to `xPCTarget.MaxLogSamples`, which returns the maximum number of samples that can be logged (because of buffer size constraints), `xPCtarget.NumLogSamples` returns the number of samples actually logged.

Note that the target application must be stopped before you get the number.

xPCTarget.NumLogWraps

Purpose Return number of times log buffer wraps

Prototype `long NumLogWraps();`

Member Of `XPCAPICOMLIB.xPCTarget`

Return The `xPCTarget.NumLogWraps` method returns the number of times the log buffer wraps. If the method detects an error, it returns `-1`.

Description The `xPCTarget.NumLogWraps` method returns the number of times the log buffer wraps.

Note that the target application must be stopped before you get the number.

Purpose

Change parameter value

Prototype

```
long SetParam(long paramIdx, SAFEARRAY(double)*
newparamVal);
```

Member Of

`XPCAPICOMLIB.xPCTarget`

Arguments

[in] *`paramIdx`*	Parameter index.
[in, out] *`newparamVal`*	Vector of doubles, assumed to be the size required by the parameter type.

Return

If the method detects an error, it returns `0`. Otherwise, it returns `-1`.

Description

The `xPCTarget.SetParam` method sets the parameter *`paramIdx`* to the value in *`newparamVal`*. For matrices, *`newparamVal`* should be a vector representation of the matrix in column-major format. Although *`newparamVal`* is a vector of doubles, the method converts the values to the expected data types (using truncation) before setting them.

See Also

API methods `xPCTarget.GetParam`, `xPCTarget.GetParamDims`, `xPCTarget.GetParamIdx`

xPCTarget.SetSampleTime

Purpose Change sample time for target application

Prototype `long SetSampleTime(double ts);`

Member Of `XPCAPICOMLIB.xPCTarget`

Arguments

[in] *`ts`*	Sample time for the target application.

Return If the method detects an error, it returns `0`. Otherwise, it returns `-1`.

Description The `xPCTarget.SetSampleTime` method sets the sample time, in seconds, of the target application to *`ts`*. Use this method only when the application is stopped.

Purpose Change stop time of target application

Prototype

```
long SetStopTime(double tfinal);
```

Member Of `XPCAPICOMLIB.xPCTarget`

Arguments

[in] *`tfinal`*	Enter the stop time, in seconds.

Return If the method detects an error, it returns `0`. Otherwise, it returns `-1`.

Description The `xPCTarget.SetStopTime` method sets the stop time of the target application to the value in *`tfinal`*. The target application will run for this number of seconds before stopping. Set *`tfinal`* to `-1.0` to set the stop time to infinity.

Purpose Start target application

Prototype `long StartApp()`

Member Of `XPCAPICOMLIB.xPCTarget`

Return If the method detects an error, it returns `0`. Otherwise, it returns `-1`.

Description The `xPCTarget.StartApp` method starts the target application loaded on the target machine.

Purpose Stop target application

Prototype `long StopApp();`

Member Of `XPCAPICOMLIB.xPCTarget`

Return If the method detects an error, it returns `0`. Otherwise, it returns `-1`.

Description The `xPCTarget.StopApp` method stops the target application loaded on the target computer. The target application remains loaded, and the parameter changes you made remain intact. If you want to stop and unload an application, use `xPCTarget.UnLoadApp`.

Purpose

Unload target application

Prototype

`long UnLoadApp();`

Member Of

`XPCAPICOMLIB.xPCTarget`

Return

If the method detects an error, it returns `0`. Otherwise, it returns `-1`.

Description

The `xPCTarget.UnloadApp` method stops the current target application, removes it from the target computer memory, and resets the target computer in preparation for receiving a new target application. The method `xPCTarget.LoadApp` calls this method before loading a new target application.

8

MATLAB API

MATLAB API — Alphabetical List

Purpose

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.

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.

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
```

Purpose

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.

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`

Examples

Simple

```
macaddr('01:23:45:67:89:ab')

ans =

     1    35    69   103   137   171
```

See Also

“Model-Based Ethernet Communications”

profile_xpc

Purpose

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.

Input Arguments

profileInfo - Profile configuration information
structure

Profile configuration data, consisting of the following fields:

rawdataonhost - Flag specifying whether the raw data is on host 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 host 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 host computer monitor.

- `1` — Do not display the execution profile plot on the host computer monitor.

Data Types
`double`

noreport - Flag suppressing code execution profiling report

`0` (default) | `1`

- `0` — Display the code execution profiling report on the host computer monitor.
- `1` — Do not display the code execution profiling report on the host computer monitor.

Data Types
`double`

Output Arguments

profData - Profile results data

structure

Profile results data stored in an object of type `coder.profile.ExecutionTime`. The same data is assigned to the variable declared in the Configuration Parameters **Workspace variable** text box.

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`.

Examples

Concurrent Execution Example

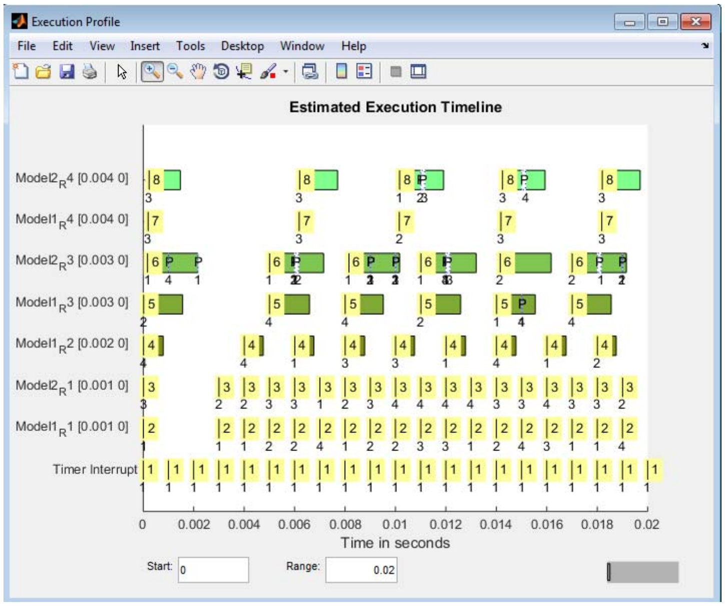
Profile the concurrent execution model `dxpcmds6t` using default settings on a multicore target computer.

Configure model `dxpcmds6t` for profiling. Build, download, and execute the model.

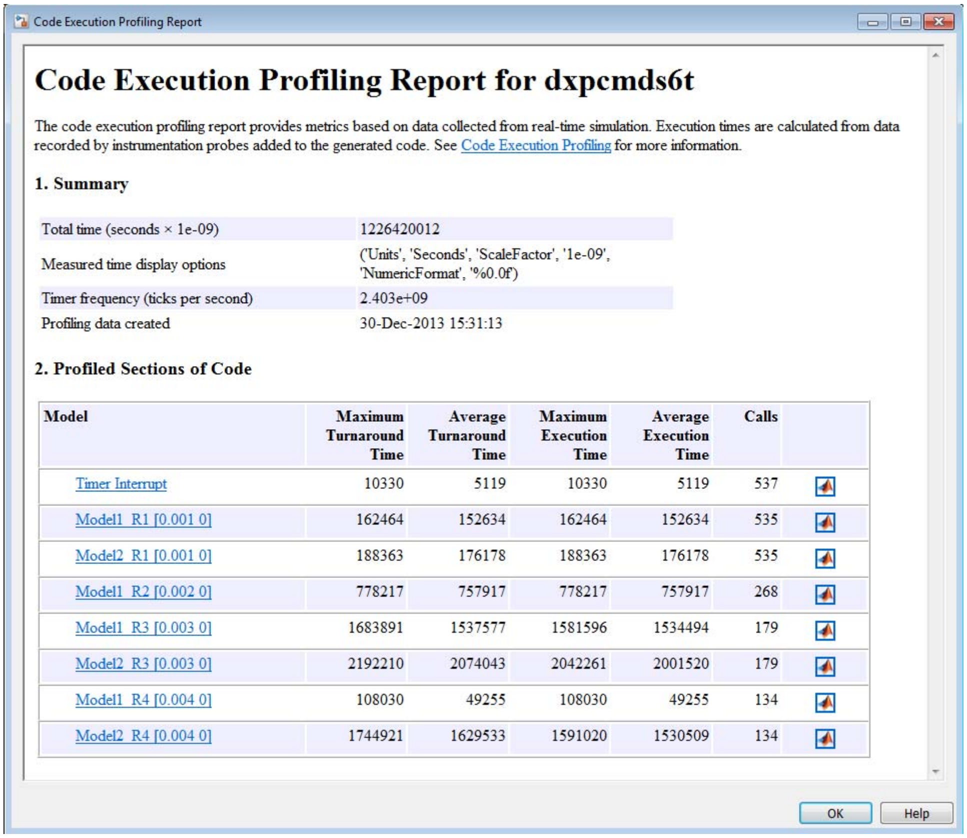
Profile the target application execution.

```
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.



The Code Execution Profiling Report displays model execution profile results for each task.


Code Execution Profiling Report

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	

OK Help

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.

See Also

`TimerTicksPerSecond` | `Sections`

Related Examples

- "Configure Target Application for Profiling"
- "Generate Target Application Execution Profile"

slrt

Purpose

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.

`target_object = slrt(target_name)` constructs a target object representing the target computer designated by `target_name`.

Input Arguments

target_name - Name assigned to target computer
string

Example: 'TargetPC1'

Data Types
`char`

Output Arguments

target_object - Target object representing target computer
structure

Examples

Default target computer

Creates a target object to communicate with the default target computer. Reports the status of the default target computer, in this case connected with the loader running.

```
target_object = slrt

Target: TargetPC1
   Connected            = Yes
   Application          = loader
```

Specific target computer

Creates a target object to communicate with target computer `TargetPC1`, Reports the status of the target computer, in this case not connected.

```
target_object = slrt('TargetPC1')

Target: TargetPC1
   Connected            = No
```

See Also

SimulinkRealTime.target (constructor) | SimulinkRealTime.TargetSettings | SimulinkRealTime.target.get | SimulinkRealTime.target.set

Purpose

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.

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 (`this`, `-reboot`) in single quotes (`'this'`,`'-reboot'`).

Example:

Data Types
`char`

Output Arguments

expected_results - Results of predefined benchmarks previously run on representative target computers
struct array

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

Tips

- Before you run `slrtbench`, you must be able to start the target computer, connect the host 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 information about the target computer interrupt latencies.

Examples

slrtbench

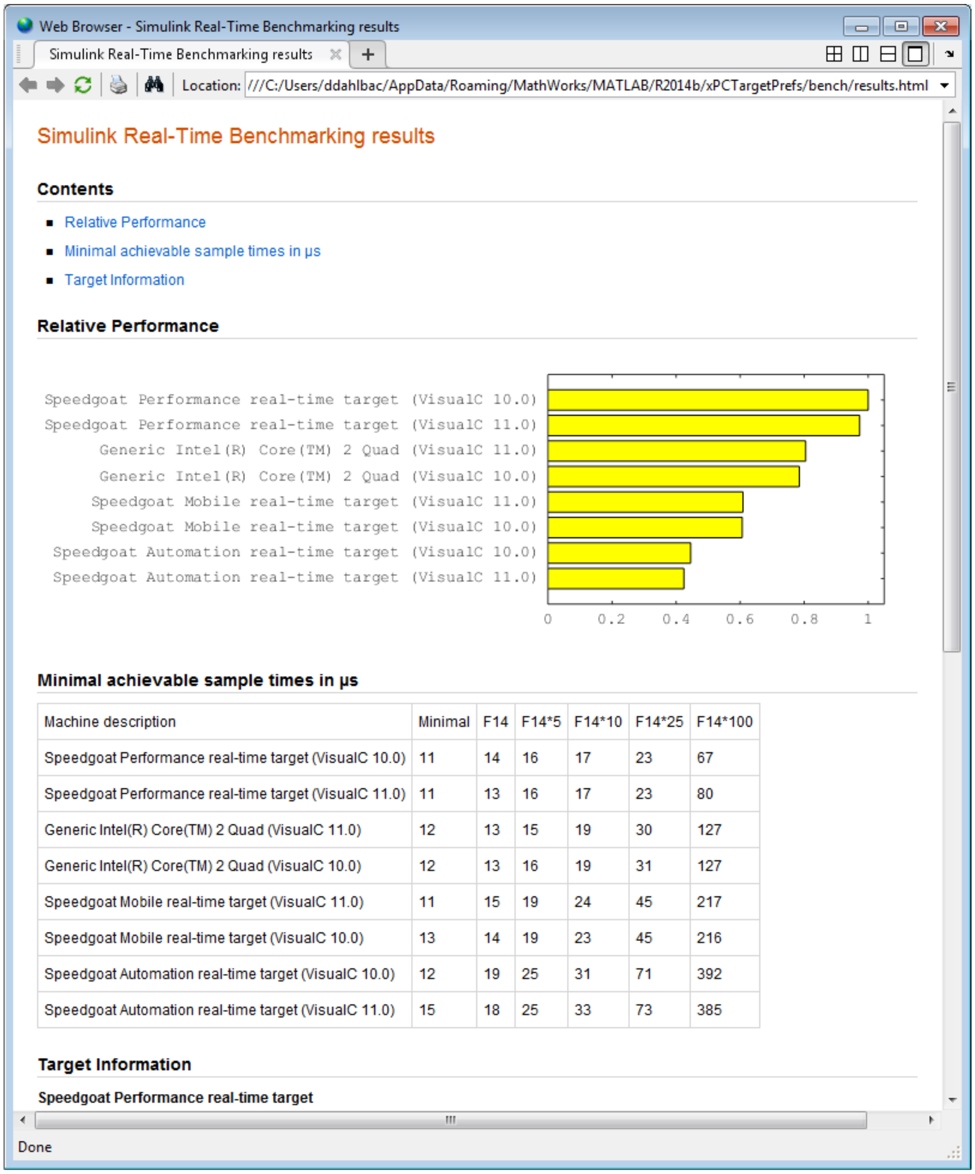
Show representative benchmark results from various target computers.

Start the target computer and run confidence test.

```
slrttest
```

Display representative results on predefined benchmarks.

```
slrtbench
```


Web Browser - Simulink Real-Time Benchmarking results

Location: ///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 (VisualC 10.0)
Speedgoat Performance real-time target (VisualC 11.0)
Generic Intel(R) Core(TM) 2 Quad (VisualC 11.0)
Generic Intel(R) Core(TM) 2 Quad (VisualC 10.0)
Speedgoat Mobile real-time target (VisualC 11.0)
Speedgoat Mobile real-time target (VisualC 10.0)
Speedgoat Automation real-time target (VisualC 10.0)
Speedgoat Automation real-time target (VisualC 11.0)
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 (VisualC 10.0)	11	14	16	17	23	67
Speedgoat Performance real-time target (VisualC 11.0)	11	13	16	17	23	80
Generic Intel(R) Core(TM) 2 Quad (VisualC 11.0)	12	13	15	19	30	127
Generic Intel(R) Core(TM) 2 Quad (VisualC 10.0)	12	13	16	19	31	127
Speedgoat Mobile real-time target (VisualC 11.0)	11	15	19	24	45	217
Speedgoat Mobile real-time target (VisualC 10.0)	13	14	19	23	45	216
Speedgoat Automation real-time target (VisualC 10.0)	12	19	25	31	71	392
Speedgoat Automation real-time target (VisualC 11.0)	15	18	25	33	73	385

Target Information

Speedgoat Performance real-time target

Done

slrtbench this

Benchmark the target computer with the predefined benchmarks.

Start the target computer and run confidence test.

```
slrttest
```

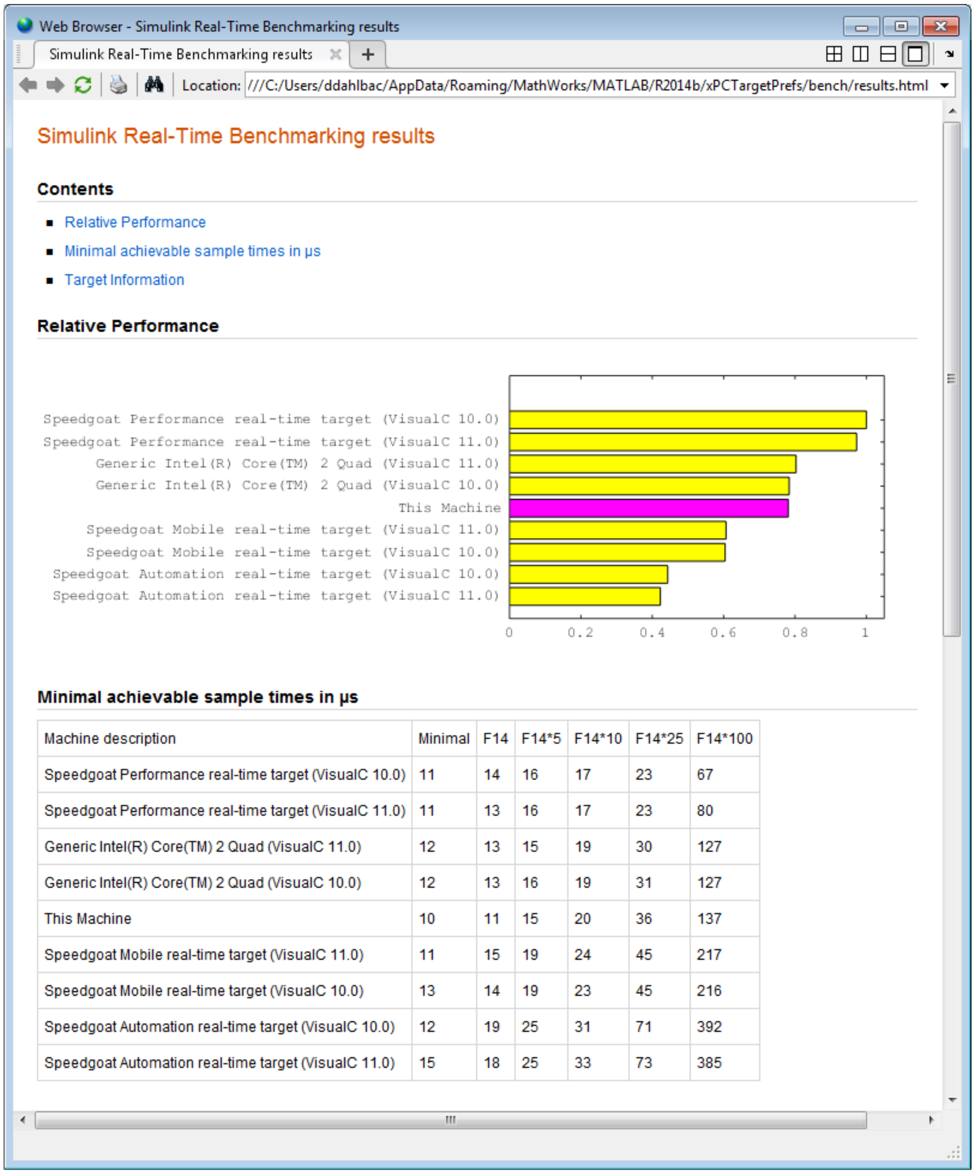
Run the benchmark models and display results.

```
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: ///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 (VisualC 10.0)
Speedgoat Performance real-time target (VisualC 11.0)
Generic Intel(R) Core(TM) 2 Quad (VisualC 11.0)
Generic Intel(R) Core(TM) 2 Quad (VisualC 10.0)
This Machine
Speedgoat Mobile real-time target (VisualC 11.0)
Speedgoat Mobile real-time target (VisualC 10.0)
Speedgoat Automation real-time target (VisualC 10.0)
Speedgoat Automation real-time target (VisualC 11.0)
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 (VisualC 10.0)	11	14	16	17	23	67
Speedgoat Performance real-time target (VisualC 11.0)	11	13	16	17	23	80
Generic Intel(R) Core(TM) 2 Quad (VisualC 11.0)	12	13	15	19	30	127
Generic Intel(R) Core(TM) 2 Quad (VisualC 10.0)	12	13	16	19	31	127
This Machine	10	11	15	20	36	137
Speedgoat Mobile real-time target (VisualC 11.0)	11	15	19	24	45	217
Speedgoat Mobile real-time target (VisualC 10.0)	13	14	19	23	45	216
Speedgoat Automation real-time target (VisualC 10.0)	12	19	25	31	71	392
Speedgoat Automation real-time target (VisualC 11.0)	15	18	25	33	73	385

slrtbench this -verbose -reboot -cleanup

Benchmark the target computer with the predefined benchmarks, and then delete build files.

Start the target computer and run confidence test.

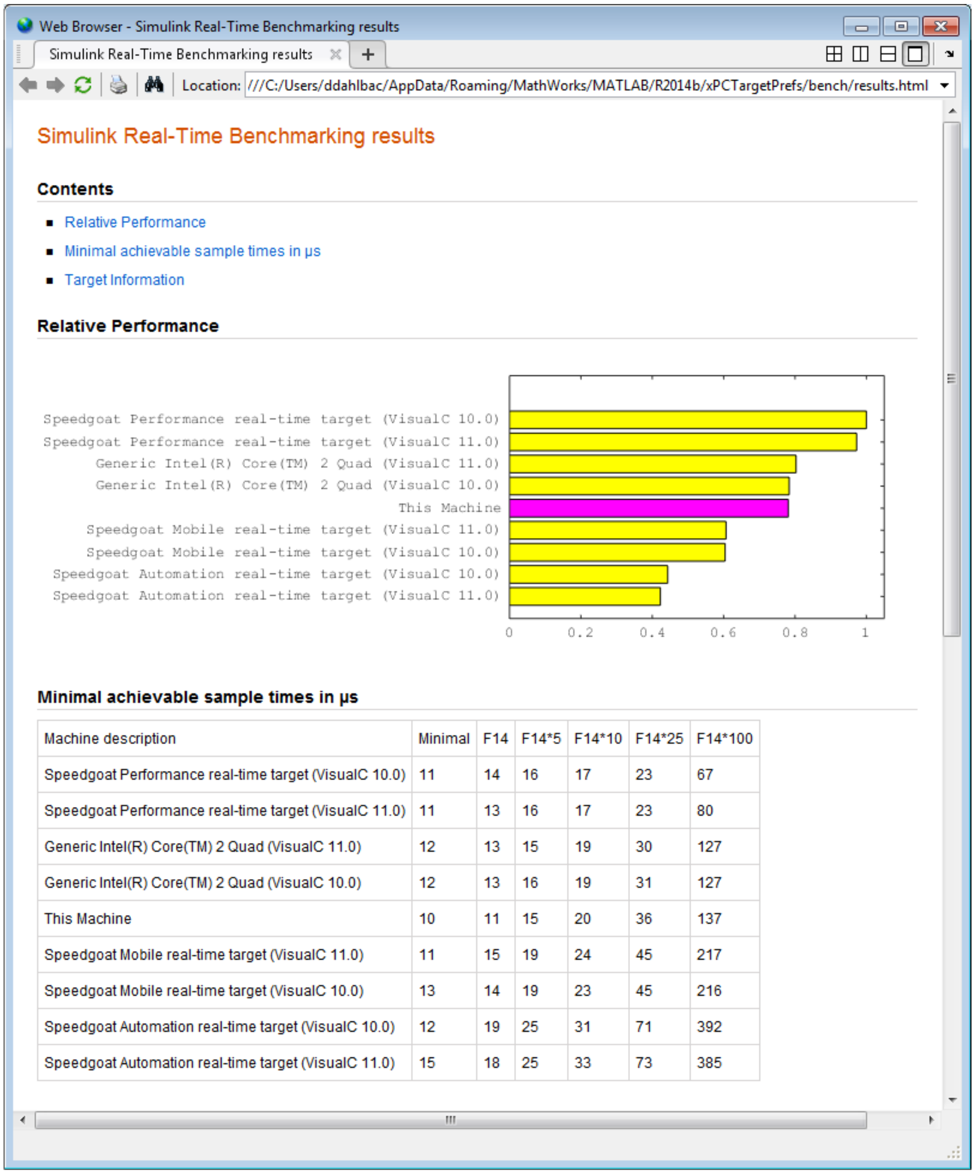
```
slrttest
```

Run the benchmark models, 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: ///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 (VisualC 10.0)
Speedgoat Performance real-time target (VisualC 11.0)
Generic Intel(R) Core(TM) 2 Quad (VisualC 11.0)
Generic Intel(R) Core(TM) 2 Quad (VisualC 10.0)
This Machine
Speedgoat Mobile real-time target (VisualC 11.0)
Speedgoat Mobile real-time target (VisualC 10.0)
Speedgoat Automation real-time target (VisualC 10.0)
Speedgoat Automation real-time target (VisualC 11.0)
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 (VisualC 10.0)	11	14	16	17	23	67
Speedgoat Performance real-time target (VisualC 11.0)	11	13	16	17	23	80
Generic Intel(R) Core(TM) 2 Quad (VisualC 11.0)	12	13	15	19	30	127
Generic Intel(R) Core(TM) 2 Quad (VisualC 10.0)	12	13	16	19	31	127
This Machine	10	11	15	20	36	137
Speedgoat Mobile real-time target (VisualC 11.0)	11	15	19	24	45	217
Speedgoat Mobile real-time target (VisualC 10.0)	13	14	19	23	45	216
Speedgoat Automation real-time target (VisualC 10.0)	12	19	25	31	71	392
Speedgoat Automation real-time target (VisualC 11.0)	15	18	25	33	73	385

slrtbench xpcosc

Use model `xpcosc` to benchmark the target computer, 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, 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 -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.

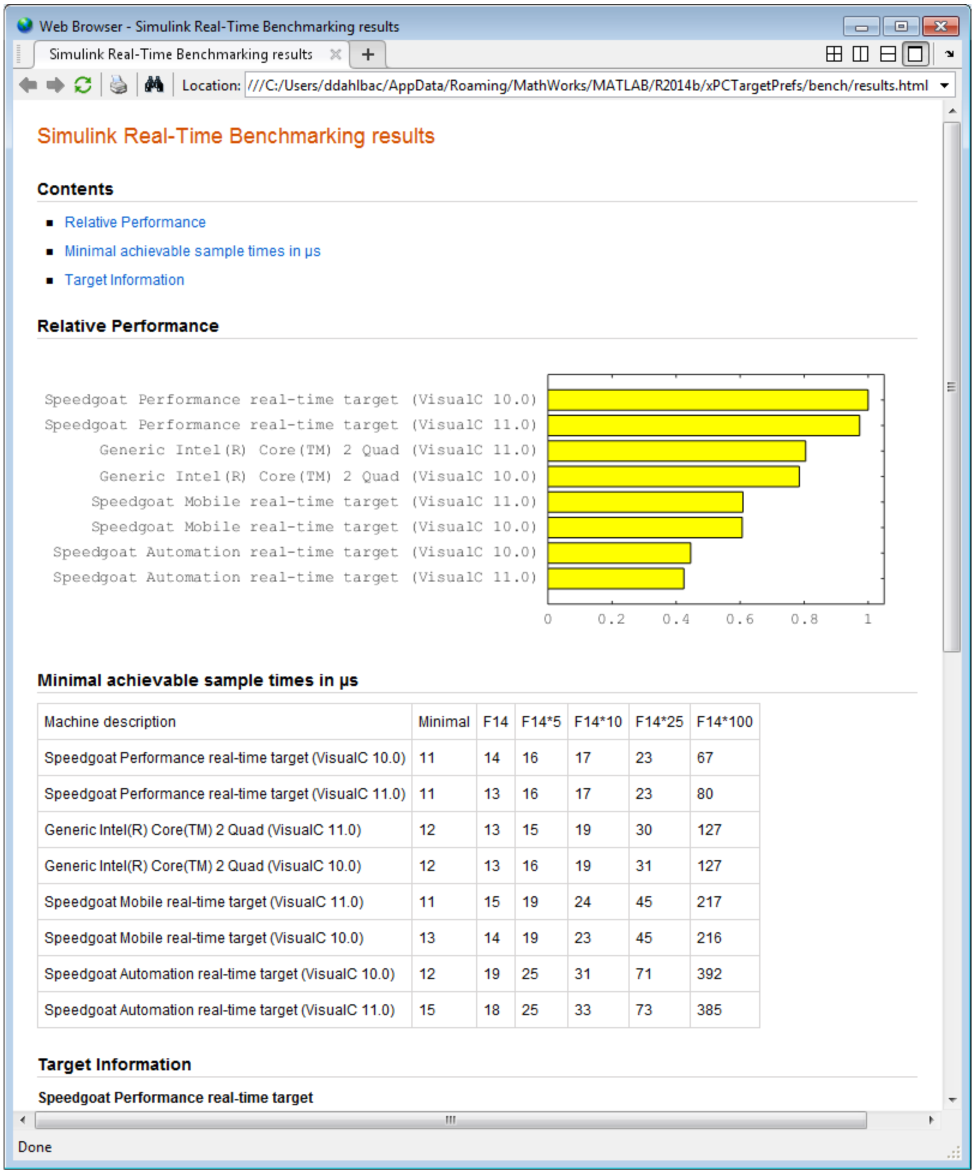
```
slrttest
```

Return an array with representative results for each processor type, in arbitrary order.

```
expected_results = slrtbench();
expected_results(1)

ans =

         Machine: 'Generic Intel(R) Core(TM) 2 Quad (VisualC 10.0)'
    BenchResults: [1.2359e-05 1.3184e-05 1.5623e-05 1.8978e-05
                   3.1175e-05 1.2723e-04]
            Desc: '%  Intel(R) Core(TM)2 Quad CPU Q6600 @ 2.40GHz
%  RAM: 2044MB
%  CP...'
```



Web Browser - Simulink Real-Time Benchmarking results
Simulink Real-Time Benchmarking results
Location: ///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 (VisualC 10.0)
Speedgoat Performance real-time target (VisualC 11.0)
Generic Intel(R) Core(TM) 2 Quad (VisualC 11.0)
Generic Intel(R) Core(TM) 2 Quad (VisualC 10.0)
Speedgoat Mobile real-time target (VisualC 11.0)
Speedgoat Mobile real-time target (VisualC 10.0)
Speedgoat Automation real-time target (VisualC 10.0)
Speedgoat Automation real-time target (VisualC 11.0)
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 (VisualC 10.0)	11	14	16	17	23	67
Speedgoat Performance real-time target (VisualC 11.0)	11	13	16	17	23	80
Generic Intel(R) Core(TM) 2 Quad (VisualC 11.0)	12	13	15	19	30	127
Generic Intel(R) Core(TM) 2 Quad (VisualC 10.0)	12	13	16	19	31	127
Speedgoat Mobile real-time target (VisualC 11.0)	11	15	19	24	45	217
Speedgoat Mobile real-time target (VisualC 10.0)	13	14	19	23	45	216
Speedgoat Automation real-time target (VisualC 10.0)	12	19	25	31	71	392
Speedgoat Automation real-time target (VisualC 11.0)	15	18	25	33	73	385

Target Information

Speedgoat Performance real-time target

current_results = slrtbench('xpcosc','-verbose','-reboot','-cleanup')

Benchmark the target computer using the `xpcosc` model and all control options, and return a structure array with results.

Start the target computer and run confidence test.

```
slrttest
```

Build `'xpcosc'`, print build messages, run benchmark, delete build files, restart the target computer, 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
```

See Also

slrttest

External Web Sites

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

Purpose

Construct skeleton for custom driver

Syntax

```
slrtdrivertool
```

Description

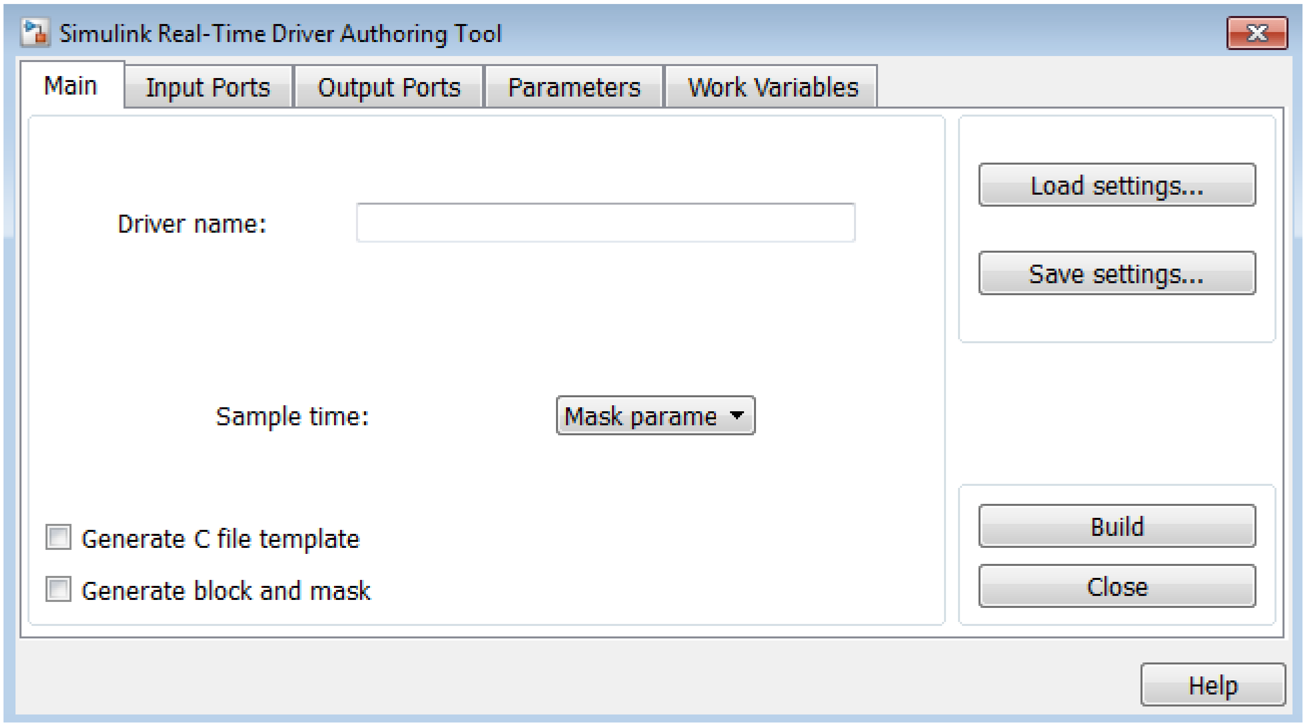
`slrtdrivertool` opens the Simulink Real-Time Driver Authoring Tool. Using this tool, you can:

- 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

Purpose

Configure target computer and target application for execution

Syntax

```
slrtexplr
```

Description

The command `slrtexplr` opens Simulink Real-Time Explorer, providing the following capabilities:

- Environment configuration — Use the **Target Properties** pane to configure the Simulink Real-Time environment properties and create a Simulink Real-Time bootable image.

  Use node **File system** under the **MATLAB Session** tree to browse the target computer file system.

- Control — Use the **Targets** and **Applications** panes to load, unload, and run target applications. You can change stop time and sample times without regenerating code. You can get task execution time information during or after the last run.
- Signal acquisition — Use the **Scopes** pane and the **Model Hierarchy** node in the **Applications** pane to interactively monitor signals, add a host, target, or file scope, add or remove signals, and save and load signal groups.
- Parameter tuning — Use the **Model Hierarchy** node in the **Applications** pane to change tunable parameters in your target application and save and load parameter groups.
- Instrumentation — Use the **Palette** and **Panels** panes to create a graphical instrument panel for acquiring signals and tuning parameters.
- Window configuration — Use the tab and the icon to make multiple workspaces visible simultaneously.

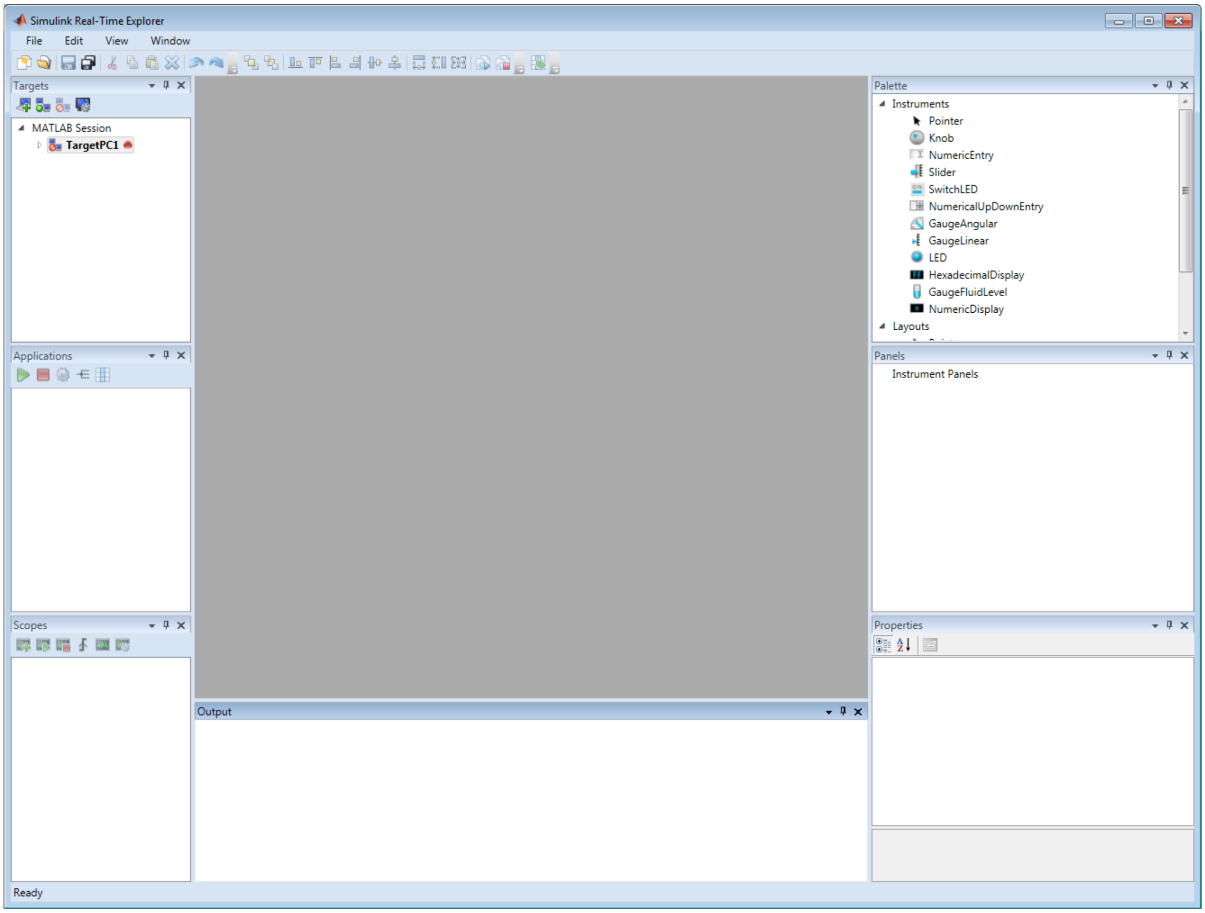
  Use **File > Save Layout** and **Load Layout** to save and restore the Simulink Real-Time Explorer window layout.

Examples

Default

Open Simulink Real-Time Explorer

```
slrtexplr
```



Related Examples

- "Ethernet Communication Setup"
- "RS-232 Communication Setup"
- "Target Computer Settings"
- "Target Boot Methods"
- "Execute Target 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"

slrtgetCC

Purpose

Compiler settings for host 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 host computer.

`[compilers] = slrtgetCC('installed')` returns in a structure the supported compilers installed on the host computer.

Output Arguments

type - Type of compiler
`VisualC`

Simulink Real-Time supports the Microsoft Visual Studio C compiler only.

location - Folder path to compiler on host computer
string

compilers - Array of structures containing compiler type, name, and location
array of structures

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'
```

See Also

`slrtsetCC` | `mex`

External Web Sites

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

slrtpingtarget

Purpose

Test communication between host 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 host and target computers. Otherwise, returns `failed`.

`slrtpingtarget` without an argument returns `success` if the host 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 host computer can communicate with target computer `target_computer_name` using the settings for that target computer. Otherwise, returns `failed`.

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`

Examples

Check communication with default target computer

```
slrtpingtarget
```

Check communication with specified target computer

```
slrtpingtarget TargetPC1
```

slrtsetCC

Purpose

Compiler settings for host computer environment

Syntax

```
slrtsetCC('setup')
slrtsetCC('type','location')
```

Description

`slrtsetCC('setup')` queries the host 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.

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 host computer

string

Data Types
`char`

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')
```

See Also

slrtgetCC | mex

External Web Sites

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

slrttest

Purpose

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 host and target computers.
- Restart the target computer to reset the target environment.
- Build a target application on the host computer.
- Download a target application to the target computer.
- Check communication between the host and target computers using commands.
- Execute a target application.
- Compare the results of a simulation and the target application run.

`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`.

Input Arguments

target_name - Specifies target name
string

The target name string is case sensitive.

Example: `'TargetPC1'`

Examples

Test default target computer

Target computer must be running and physically connected to the host computer.

```
slrttest
```

Test default target computer, skipping reboot test

Target computer must be running and physically connected to the host computer.

```
slrttest('noreboot')
```

Test specified target computer, skipping reboot test

Target computer must be running and physically connected to the host computer.

```
slrttest('TargetPC1','noreboot')
```

Concepts

- “Troubleshooting in Simulink Real-Time”

SimulinkRealTime.addTarget

Purpose

Add new Simulink Real-Time target object

Syntax

`env_object.Add`

Description

`SimulinkRealTime.addTarget` is a package method of `SimulinkRealTime`. It creates an object on the host computer that represent the target computer.

Examples

Add a new Simulink Real-Time target object (`tgs`) to the system. The `get(tgs)` function calls return the number of target computers before and after you add a target computer.

```
tgs=SimulinkRealTime.target;
get(tgs);
tgs.Add;
get(tgs);
```

See Also

`SimulinkRealTime.targetSettings.set` | `SimulinkRealTime.targetSettings.get`

Purpose

Copy file from target computer to host 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 host computer.

`SimulinkRealTime.copyFileToHost(target_obj,file_name)` copies file `file_name` from the target computer represented by `target_obj` to the host computer.

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 host computer.

Example: `'myFile.txt'`

Example: `'c:\subDir\myFile.txt'`

Data Types
`char`

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')
```

See Also

`SimulinkRealTime.copyFileToTarget` | `SimulinkRealTime.fileSystem.cd` | `SimulinkRealTime.fileSystem.dir` | `SimulinkRealTime.fileSystem.pwd`

Purpose

Copy file from host 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 host computer to the default target computer.

`SimulinkRealTime.copyFileToTarget(target_obj,file_name)` copies file `file_name` from the host computer to the target computer represented by `target_obj`.

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 host computer

`file name string | full path name string`

The file being copied must exist in the current folder on the host 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 host 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`

Examples

Copy File to Default Target Computer Top Folder

Copy file from current folder on host 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 host computer to full path location on target computer `TargetPC1`.

```
tg = slrt('TargetPC1');
SimulinkRealTime.copyFileToTarget(tg,'c:\xpcosc\data1.dat')
```

See Also

`SimulinkRealTime.copyFileToHost` | `SimulinkRealTime.fileSystem.cd` | `SimulinkRealTime.fileSystem.dir` | `SimulinkRealTime.fileSystem.pwd`

Purpose

Create Simulink Real-Time boot disk or DOS Loader files

Syntax

```
SimulinkRealTime.createBootImage
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_object)`creates a boot image for the target computer indicated by `target_object`, which can be the name of a target computer or 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 host 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 target application. `SimulinkRealTime.createBootImage` does not generate a standalone application.

Use `SimulinkRealTime.targetSettings.set` to update the environment properties. 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.targetSettings.set` | `SimulinkRealTime.getTargetSettings`

How To

- “Target Boot Methods”
- “Command-Line Target Boot Methods”

SimulinkRealTime.getSupportInfo

Purpose

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.

SimulinkRealTime.getTargetSettings

Purpose

Display specific target computer environment object

Syntax

```
SimulinkRealTime.getTargetSettings
SimulinkRealTime.getTargetSettings(env_object_name)
env = SimulinkRealTime.getTargetSettings( ___ )
```

Description

`SimulinkRealTime.getTargetSettings` displays the environment object representing the default computer.

`SimulinkRealTime.getTargetSettings(env_object_name)` displays the environment object representing a particular target computer.

`env = SimulinkRealTime.getTargetSettings( ___ )` returns the environment object representing the target computer. Access the environment properties using the `SimulinkRealTime.targetSettings.get` and `SimulinkRealTime.targetSettings.set` functions.

Examples

Display the default target environment object.

```
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
```

Retrieve a target environment object for a specific target computer. Use it to access a property.

```
env = SimulinkRealTime.getTargetSettings('TargetPC1');
env.get('HostTargetComm')
```

See Also

SimulinkRealTime.targetSettings.set | SimulinkRealTime.targetSettings.get

SimulinkRealTime.pingTarget

Purpose

Test communication between host 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 host and target computers. Otherwise, returns `failed`.

`SimulinkRealTime.pingTarget` without an argument returns `success` if the host 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 host 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'`).

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`

Examples

Check communication with default target computer

```
SimulinkRealTime.pingTarget
```

Check communication with specified target computer

```
SimulinkRealTime.pingTarget('TargetPC1')
```

SimulinkRealTime.removeTarget

Purpose

Remove environment data associated with target name

Syntax

`SimulinkRealTime.removeTarget('target_name')`

Description

Method of package `SimulinkRealTime` . `SimulinkRealTime.removeTarget` removes the definitions and settings for the indicated target from the system, invalidating the target objects associated with that target. 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.targetSettings.set` | `SimulinkRealTime.targetSettings.get`

SimulinkRealTime.utils.bytes2file

Purpose

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]`.

At every sample time, set up the real-time From File block to output:

```
28 bytes
(1 * sizeof('single') + 3 * sizeof('double'))
```

SimulinkRealTime.utils.createInstrumentationModel

Purpose

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 target application.

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`

Examples

Generate an interface model

```
SimulinkRealTime.utils.createInstrumentationModel('xpcosc')
```

Purpose

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 host 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, copy the data from the target computer using the `SimulinkRealTime.fileSystem.fread` method.

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`.

Examples

These examples access a file on a target computer using different methods and plot the results. The model includes one scalar signal connected to a real-time Scope block of type `File`. The model has been built, downloaded, and run, producing file `'data.dat'` on the target computer.

Using `slrtfile_name` argument to read file and plot results

Upload the file using `SimulinkRealTime.fileSystem` methods. Read the file on the host using `SimulinkRealTime.utils.getFileScopeData`. Plot the results.

Upload file `'data.dat'` from the target computer.

```
fs = SimulinkRealTime.fileSystem;
fs.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 the file on the target computer using `SimulinkRealTime.fileSystem` methods. Store the data in a workspace variable. Convert the data to MATLAB format using `SimulinkRealTime.utils.getFileScopeData`. Plot the results.

Read file `'data.dat'` from the target computer.

```
fs = SimulinkRealTime.fileSystem;
h=fs.fopen('data.dat');
slrtfile_data=fs.fread(h);
fs.fclose(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));
```

See Also

Scope | `SimulinkRealTime.fileSystem`

Purpose Store target environment properties

Description **Methods**

Method	Description
`SimulinkRealTime.targetSettings.get`	Return property values for an environment object
`SimulinkRealTime.targetSettings.set`	Change property values for an environment object

Properties

The environment properties define communication between the host computer and target computer and the type of target boot image created during the setup process. An understanding of the environment properties helps you configure the Simulink Real-Time environment.

To access target environment properties from the Command Window, use `SimulinkRealTime.targetSettings.get` and `SimulinkRealTime.targetSettings.set`.

To access the environment properties 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**.

The environment properties for a target environment object are listed in the following tables.

- Host-to-Target communication on page 66
- Target settings on page 72
- Boot configuration on page 76

Host-to-Target communication

Environment Property	Description
`HostTargetComm`	MATLAB property values are `'RS232'` and `'TcpIp'`. From 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`.

Environment Property	Description
	If you select TCP/IP, then you must set the other properties that start with TcpIp. **Note** RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.
RS232Baudrate	MATLAB property values are '115200', '57600', '38400', '19200', '9600', '4800', '2400', and '1200'. From the Simulink Real-Time Explorer **Baud rate** list, select one of 1200, 2400, 4800, 9600, 19200, 38400, 57600, or 115200.
RS232HostPort	MATLAB property values are 'COM1' and 'COM2'. From the Simulink Real-Time Explorer **Host port** list, select one of COM1 or COM2. The software determines the COM port on the target computer. Before you can select an RS-232 port, you must set the HostTargetComm property to RS232.

Environment Property	Description
`TcpIpGateway`	MATLAB property value is `'xxx.xxx.xxx.xxx'`. In the Simulink Real-Time Explorer **Gateway** box, type the IP address for your gateway. This property is set by default to `255.255.255.255`, which means that a gateway is not used to connect to the target computer. If you communicate with your target computer from within a LAN that uses gateways, and your host and target computers are connected through a gateway, you must enter a value for this property. If your LAN does not use gateways, you do not need to change this property. Consult your system administrator for this value.
`TcpIpSubNetMask`	MATLAB property value is `'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. For example, `255.255.255.0`.

Environment Property	Description
TcpIpTargetAddress	MATLAB property value is '*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. For example, 192.168.0.10.
TcpIpTargetBusType	MATLAB property values are 'PCI', 'ISA', and 'USB'. From the Simulink Real-Time Explorer **Bus type** list, select one of PCI, ISA, or USB. This property is set by default to PCI. It determines the bus type of your target computer. You do not need to define a bus type for your host 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.

Environment Property	Description
`TcpIpTargetDriver`	MATLAB property values are `'3C90x'`, `'I8254x'`, `'I82559'`, `'NE2000'`, `'NS83815'`, `'R8139'`, `'R8168'`, `'Rhine'`, `'RTLANCE'`, `'SMC91C9X'`, `'USBAX772'`, `'USBAX172'`, and `'Auto'`. From 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`.
`TcpIpTargetISAIRQ`	MATLAB property value is `'`*`n`*`'`. *`n`* is between 5 and 15 inclusive. From the Simulink Real-Time Explorer **IRQ** list, select an IRQ value. If you are using an ISA bus Ethernet card, you must enter values for the properties `TcpIpISAMemPort` and `TcpIpISAIRQ`. The values of these properties must correspond to the jumper or ROM settings on the ISA bus Ethernet card. On your ISA bus card, assign an IRQ and I/O-port base address by moving the jumpers on the card. Set the IRQ to 5, 10, or 11. If one of these hardware settings leads to a conflict in your target

Environment Property	Description
	computer, choose another IRQ and make the corresponding changes to your jumper settings.
`TcpIpTargetISAMemPort`	MATLAB property value is `'0xnnnn'`. In the Simulink Real-Time Explorer **Address** box, type an I/O port base address. If you are using an ISA bus Ethernet card, you must enter values for the properties `TcpIpISAMemPort` and `TcpIpISAIRQ`. The values of these properties must correspond to the jumper settings or ROM settings on your ISA bus Ethernet card. On your ISA bus card, assign an IRQ and I/O port base address by moving the jumpers on the card. Set the I/O port base address to 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.
`TcpIpTargetPort`	MATLAB property value is `'xxxxx'`. In the Simulink Real-Time Explorer **Port** box, type a port address greater than `20000`.

Environment Property	Description
	This property is set by default to 22222. The default value is higher than the reserved area (telnet, ftp, . . .) and is used only on the target computer.

Target settings

Environment Property	Description
EthernetIndex	MATLAB property value is '*n*'. '*n*' indicates the index number for the Ethernet card on a target computer. The (n-1)th Ethernet card on the target computer has an index number 'n'. The default index number is '0'. There is no corresponding Simulink Real-Time Explorer interface element. If the target computer has multiple Ethernet cards, you must select one of the cards for host-target communication. This option returns the index number of the card selected on the target computer upon starting.
LegacyMultiCoreConfig	MATLAB property values are 'on' and 'off'. The default value is 'off'. There is no corresponding Simulink Real-Time Explorer interface element. Set this value to 'on' only if your target computer contains hardware not compliant with the Advanced Configuration and Power Interface (ACPI) standard. Otherwise, set this value to 'off'.

Environment Property	Description
`MaxModelSize`	Supported MATLAB property values are `'1MB'` and `'4MB'`. The default value is `'1MB'`. Value `'16MB'` is not supported. From 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. Choosing the maximum model size reserves the specified amount of memory on the target computer for the target application. Memory not used by the target application is used by the kernel and by the heap for data logging. Selecting too high a value leaves less memory for data logging. Selecting too low a value does not reserve enough memory for the target application and creates an error. You can approximate the size of the target application by the size of the DLM file produced by the build process.
`MulticoreSupport`	MATLAB property values are `'on'` and `'off'`. The default value is `'off'`. If your target computer has multicore processors, select the Simulink Real-Time Explorer **Multicore CPU** check box to take advantage of these processors for background tasks. Otherwise, clear it.
`Name`	MATLAB property is the target computer name string. 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.

Environment Property	Description
`NonPentiumSupport`	MATLAB property values are `'on'` and `'off'`. The default value is `'off'`. 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. If your target computer has a Pentium or higher compatible processor, selecting this check box slows the performance of your target computer.
`SecondaryIDE`	MATLAB property values are `'on'` and `'off'`. The default value is `'off'`. If you want to use the disks connected to a secondary IDE controller, select the Simulink Real-Time Explorer **Secondary IDE** check box. Otherwise, clear it.
`ShowHardware`	MATLAB property values are `'on'` and `'off'`. The default value is `'off'`. There is no corresponding Simulink Real-Time Explorer interface element. 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 host computer cannot communicate with the target computer after the kernel starts with `ShowHardware` set.

Environment Property	Description
`TargetRAMSizeMB`	MATLAB property values are `'Auto'` and `'xxx'`. *`xxx`* is a positive value specifying the total amount of RAM, in megabytes, installed on the target computer. Target computer RAM is used for the kernel, target application, data logging, and other functions that use the heap. The default value is `'Auto'`. To allow the target application to read the target computer BIOS and determine the amount of memory up to a maximum of 2 GB, in Simulink Real-Time Explorer, click **RAM size Auto**. If the target 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 2 GB of memory.
`TargetScope`	MATLAB property values are `'Disabled'` and `'Enabled'`. The default value is `'Enabled'`. To display scope information graphically, set the Simulink Real-Time Explorer **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.
`USBSupport`	MATLAB property values are `'on'` and `'off'`. The default value is `'on'`. To use a USB port on the target computer, for example to connect a USB mouse, select the Simulink Real-Time Explorer **USB Support** check box. Otherwise, clear it.

Boot configuration

Environment Property	Description
`BootFloppyLocation`	Drive name for creation of target boot disk.
`DOSLoaderLocation`	Location of DOS Loader files to start target computers from devices other than floppy disk or CD.
`TargetBoot`	MATLAB property values are `'BootFloppy'`, `'CDBoot'`, `'DOSLoader'`, `'NetworkBoot'`, and `'StandAlone'`. To specify a boot mode, from the Simulink Real-Time Explorer **Boot mode** list, select one of `Removable Disk`, `CD`, `DOS Loader`, `Network`, or `Stand Alone`. To create a bootable image for the specified boot mode, click **Create boot disk**.
`TargetMACAddress`	MATLAB property value is the physical target computer MAC address from which to accept start requests when starting within a dedicated network. Format the MAC address as six pairs of hexadecimal numbers, separated by colons: `xx:xx:xx:xx:xx:xx` To update the MAC address in Simulink Real-Time Explorer,

Environment Property	Description
	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.

SimulinkRealTime.targetSettings.get

Purpose Value of target environment property

Syntax

```
property_value = env_object.property_name
property_value = env_object.get('property_name')
property_value = get(env_object,'property_name')
property_value = env_object.get
property_value = get(env_object)
```

Arguments

`env_object`	Name of a target environment object.
`property_name`	Name of a target environment object property.

Description

`property_value = env_object.property_name` gets the current value of property `property_name` from target environment object `env_object`. Alternative syntaxes are:

`property_value = env_object.get('property_name')`

`property_value = get(env_object,'property_name')`

`property_value = env_object.get` gets the values of all properties of target environment object `env_object`. An alternative syntax is:

`property_value = get(env_object)`

Get an individual environment object with the `SimulinkRealTime.getTargetSettings` method. For example:

```
tgs=SimulinkRealTime.target;
env_object=tgs.Item('TargetPC1');
property_value=env_object.HostTargetComm
```

To access the environment properties 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**.

The environment properties for a target environment object are listed in the following tables.

- "Host-to-Target Communication" on page 8-79
- "Target Settings" on page 8-84
- "Boot Configuration" on page 8-88

Host-to-Target Communication

Environment Property	Description
`HostTargetComm`	MATLAB property values are `'RS232'` and `'TcpIp'`. From 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`. **Note** RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.
`RS232Baudrate`	MATLAB property values are `'115200'`, `'57600'`, `'38400'`, `'19200'`, `'9600'`, `'4800'`, `'2400'`, and `'1200'`.

Environment Property	Description
	From the Simulink Real-Time Explorer **Baud rate** list, select one of `1200`, `2400`, `4800`, `9600`, `19200`, `38400`, `57600`, or `115200`.
`RS232HostPort`	MATLAB property values are `'COM1'` and `'COM2'`. From the Simulink Real-Time Explorer **Host port** list, select one of `COM1` or `COM2`. The software determines the COM port on the target computer. Before you can select an RS-232 port, you must set the `HostTargetComm` property to `RS232`.
`TcpIpGateway`	MATLAB property value is `'`*`xxx.xxx.xxx.xxx`*`'`. In the Simulink Real-Time Explorer **Gateway** box, type the IP address for your gateway. This property is set by default to `255.255.255.255`, which means that a gateway is not used to connect to the target computer. If you communicate with your target computer from within a LAN that uses gateways, and your host and target computers are connected through a gateway, you must enter a value for this property. If your LAN does not use gateways, you do not need

Environment Property	Description
	to change this property. Consult your system administrator for this value.
`TcpIpSubNetMask`	MATLAB property value is `'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. For example, `255.255.255.0`.
`TcpIpTargetAddress`	MATLAB property value is `'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. For example, `192.168.0.10`.
`TcpIpTargetBusType`	MATLAB property values are `'PCI'`, `'ISA'`, and `'USB'`. From the Simulink Real-Time Explorer **Bus type** list, select one of `PCI`, `ISA`, or `USB`. This property is set by default to `PCI`. It determines the bus type of your target computer. You do not need to define a bus type for your host computer. If `TcpIpTargetBusType` is set to `PCI`, then the properties

Environment Property	Description
	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.
TcpIpTargetDriver	MATLAB property values are '3C90x', 'I8254x', 'I82559', 'NE2000', 'NS83815', 'R8139', 'R8168', 'Rhine', 'RTLANCE', 'SMC91C9X', 'USBAX772', 'USBAX172', and 'Auto'. From 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.
TcpIpTargetISAIRQ	MATLAB property value is '*n*'. *n* is between 5 and 15 inclusive. From the Simulink Real-Time Explorer **IRQ** list, select an IRQ value. If you are using an ISA bus Ethernet card, you must enter values for the properties TcpIpISAMemPort and TcpIpISAIRQ. The values of these properties must correspond to the

Environment Property	Description
	jumper or ROM settings on the ISA bus Ethernet card. On your ISA bus card, assign an IRQ and I/O-port base address by moving the jumpers on the card. Set the IRQ to 5, 10, or `11`. If one of these hardware settings leads to a conflict in your target computer, choose another IRQ and make the corresponding changes to your jumper settings.
`TcpIpTargetISAMemPort`	MATLAB property value is `'0xnnnn'`. In the Simulink Real-Time Explorer **Address** box, type an I/O port base address. If you are using an ISA bus Ethernet card, you must enter values for the properties `TcpIpISAMemPort` and `TcpIpISAIRQ`. The values of these properties must correspond to the jumper settings or ROM settings on your ISA bus Ethernet card. On your ISA bus card, assign an IRQ and I/O port base address by moving the jumpers on the card. Set the I/O port base address to a value near 0x300. If one of these hardware settings leads to a conflict in your target computer, choose another I/O

Environment Property	Description
	port base address and make the corresponding changes to your jumper settings.
`TcpIpTargetPort`	MATLAB property value is `'xxxxx'`. In the Simulink Real-Time Explorer **Port** box, type a port address greater than `20000`. This property is set by default to `22222`. The default value is higher than the reserved area (`telnet`, `ftp`, . . .) and is used only on the target computer.

Target Settings

Environment Property	Description
`EthernetIndex`	MATLAB property value is `'n'`. `'n'` indicates the index number for the Ethernet card on a target computer. The `(n-1)th` Ethernet card on the target computer has an index number `'n'`. The default index number is `'0'`. There is no corresponding Simulink Real-Time Explorer interface element. If the target computer has multiple Ethernet cards, you must select one of the cards for host-target communication. This option returns the index number of the card selected on the target computer upon starting.
`LegacyMultiCoreConfig`	MATLAB property values are `'on'` and `'off'`. The default value is `'off'`.

Environment Property	Description
	There is no corresponding Simulink Real-Time Explorer interface element. Set this value to `'on'` only if your target computer contains hardware not compliant with the Advanced Configuration and Power Interface (ACPI) standard. Otherwise, set this value to `'off'`.
`MaxModelSize`	Supported MATLAB property values are `'1MB'` and `'4MB'`. The default value is `'1MB'`. Value `'16MB'` is not supported. From 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. Choosing the maximum model size reserves the specified amount of memory on the target computer for the target application. Memory not used by the target application is used by the kernel and by the heap for data logging. Selecting too high a value leaves less memory for data logging. Selecting too low a value does not reserve enough memory for the target application and creates an error. You can approximate the size of the target application by the size of the DLM file produced by the build process.
`MulticoreSupport`	MATLAB property values are `'on'` and `'off'`. The default value is `'off'`. If your target computer has multicore processors, select the Simulink Real-Time Explorer **Multicore CPU** check box to take advantage of these processors for background tasks. Otherwise, clear it.

Environment Property	Description
`Name`	MATLAB property is the target computer name string. 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.
`NonPentiumSupport`	MATLAB property values are `'on'` and `'off'`. The default value is `'off'`. 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. If your target computer has a Pentium or higher compatible processor, selecting this check box slows the performance of your target computer.
`SecondaryIDE`	MATLAB property values are `'on'` and `'off'`. The default value is `'off'`. If you want to use the disks connected to a secondary IDE controller, select the Simulink Real-Time Explorer **Secondary IDE** check box. Otherwise, clear it.
`ShowHardware`	MATLAB property values are `'on'` and `'off'`. The default value is `'off'`. There is no corresponding Simulink Real-Time Explorer interface element. 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 host computer cannot communicate with the target computer after the kernel starts with `ShowHardware` set.

Environment Property	Description
`TargetRAMSizeMB`	MATLAB property values are `'Auto'` and `'xxx'`. *`xxx`* is a positive value specifying the total amount of RAM, in megabytes, installed on the target computer. Target computer RAM is used for the kernel, target application, data logging, and other functions that use the heap. The default value is `'Auto'`. To allow the target application to read the target computer BIOS and determine the amount of memory up to a maximum of 2 GB, in Simulink Real-Time Explorer, click **RAM size Auto**. If the target 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 2 GB of memory.
`TargetScope`	MATLAB property values are `'Disabled'` and `'Enabled'`. The default value is `'Enabled'`. To display scope information graphically, set the Simulink Real-Time Explorer **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.
`USBSupport`	MATLAB property values are `'on'` and `'off'`. The default value is `'on'`. To use a USB port on the target computer, for example to connect a USB mouse, select the Simulink Real-Time Explorer **USB Support** check box. Otherwise, clear it.

Boot Configuration

Environment Property	Description
`BootFloppyLocation`	Drive name for creation of target boot disk.
`DOSLoaderLocation`	Location of DOS Loader files to start target computers from devices other than floppy disk or CD.
`TargetBoot`	MATLAB property values are `'BootFloppy'`, `'CDBoot'`, `'DOSLoader'`, `'NetworkBoot'`, and `'StandAlone'`. To specify a boot mode, from the Simulink Real-Time Explorer **Boot mode** list, select one of `Removable Disk`, `CD`, `DOS Loader`, `Network`, or `Stand Alone`. To create a bootable image for the specified boot mode, click **Create boot disk**.
`TargetMACAddress`	MATLAB property value is the physical target computer MAC address from which to accept start requests when starting within a dedicated network. Format the MAC address as six pairs of hexadecimal numbers, separated by colons: `xx:xx:xx:xx:xx:xx`

Environment Property	Description
	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.

See Also

`SimulinkRealTime.targetSettings.set`

Purpose

Change target environment object property values

Syntax

```
env_object.property_name = property_value
env_object.set('prop_name1,'prop_value1','prop_name2',. . .)
set(env_object,'prop_name1','prop_value1','prop_name2',. . .)
```

Arguments

`env_object`	Name of a target environment object.
`property_name`	Name of a target environment object property.
`property_value`	Value for a target environment object property. Always use quotation marks for character strings. Quotation marks are optional for numbers.

Description

`env_object.property_name = property_value` sets property `property_name` of target environment object `env_object` to `property_value`. Alternative syntaxes for one or more property-value pairs are:

`env_object.set('prop_name1,'prop_value1','prop_name2',. . .)`

`set(env_object,'prop_name1','prop_value1','prop_name2',. . .)`

Get an individual environment object with the `SimulinkRealTime.getTargetSettings` method. For example:

```
tgs=SimulinkRealTime.target;
env_object=tgs.Item('TargetPC1');
env_object.HostTargetComm='RS232'
```

Not all properties are user-writable.

To access the environment properties 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**.

The environment properties for a target environment object are listed in the following tables.

- "Host-to-Target Communication" on page 8-91
- "Target Settings" on page 8-97
- "Boot Configuration" on page 8-101

Host-to-Target Communication

Environment Property	Description
`HostTargetComm`	MATLAB property values are `'RS232'` and `'TcpIp'`. From 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`. **Note** RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.
`RS232Baudrate`	MATLAB property values are `'115200'`, `'57600'`, `'38400'`,

Environment Property	Description
	`'19200'`, `'9600'`, `'4800’`, `'2400'`, and `'1200'`. From the Simulink Real-Time Explorer **Baud rate** list, select one of `1200`, `2400`, `4800`, `9600`, `19200`, `38400`, `57600`, or `115200`.
`RS232HostPort`	MATLAB property values are `'COM1'` and `'COM2'`. From the Simulink Real-Time Explorer **Host port** list, select one of `COM1` or `COM2`. The software determines the COM port on the target computer. Before you can select an RS-232 port, you must set the `HostTargetComm` property to `RS232`.
`TcpIpGateway`	MATLAB property value is `'`*`xxx.xxx.xxx.xxx`*`'`. In the Simulink Real-Time Explorer **Gateway** box, type the IP address for your gateway. This property is set by default to `255.255.255.255`, which means that a gateway is not used to connect to the target computer. If you communicate with your target computer from within a LAN that uses gateways, and your host and target computers are connected through a gateway, you must enter a value for this

Environment Property	Description
	property. If your LAN does not use gateways, you do not need to change this property. Consult your system administrator for this value.
`TcpIpSubNetMask`	MATLAB property value is `'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. For example, `255.255.255.0`.
`TcpIpTargetAddress`	MATLAB property value is `'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. For example, `192.168.0.10`.

Environment Property	Description
TcpIpTargetBusType	MATLAB property values are 'PCI', 'ISA', and 'USB'. From the Simulink Real-Time Explorer **Bus type** list, select one of PCI, ISA, or USB. This property is set by default to PCI. It determines the bus type of your target computer. You do not need to define a bus type for your host 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.
TcpIpTargetDriver	MATLAB property values are '3C90x', 'I8254x', 'I82559', 'NE2000', 'NS83815', 'R8139', 'R8168', 'Rhine', 'RTLANCE', 'SMC91C9X', 'USBAX772', 'USBAX172', and 'Auto'. From the Simulink Real-Time Explorer **Target driver** list, select one of THREECOM_3C90x, INTEL_I8254x, INTEL_I82559, NE2000, NS83815, R8139, R8168,

Environment Property	Description
	Rhine, RTLANCE, SMC91C9X, USBAX772, USBAX172, or Auto.
TcpIpTargetISAIRQ	MATLAB property value is '*n*'. *n* is between 5 and 15 inclusive. From the Simulink Real-Time Explorer **IRQ** list, select an IRQ value. If you are using an ISA bus Ethernet card, you must enter values for the properties TcpIpISAMemPort and TcpIpISAIRQ. The values of these properties must correspond to the jumper or ROM settings on the ISA bus Ethernet card. On your ISA bus card, assign an IRQ and I/O-port base address by moving the jumpers on the card. Set the IRQ to 5, 10, or 11. If one of these hardware settings leads to a conflict in your target computer, choose another IRQ and make the corresponding changes to your jumper settings.

Environment Property	Description
`TcpIpTargetISAMemPort`	MATLAB property value is `'0xnnnn'`. In the Simulink Real-Time Explorer **Address** box, type an I/O port base address. If you are using an ISA bus Ethernet card, you must enter values for the properties `TcpIpISAMemPort` and `TcpIpISAIRQ`. The values of these properties must correspond to the jumper settings or ROM settings on your ISA bus Ethernet card. On your ISA bus card, assign an IRQ and I/O port base address by moving the jumpers on the card. Set the I/O port base address to 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.
`TcpIpTargetPort`	MATLAB property value is `'xxxxx'`. In the Simulink Real-Time Explorer **Port** box, type a port address greater than `20000`. This property is set by default to `22222`. The default value is higher than the reserved area

Environment Property	Description
	(telnet, ftp, . . .) and is used only on the target computer.

Target Settings

Environment Property	Description
EthernetIndex	MATLAB property value is '*n*'. '*n*' indicates the index number for the Ethernet card on a target computer. The (n-1)th Ethernet card on the target computer has an index number 'n'. The default index number is '0'. There is no corresponding Simulink Real-Time Explorer interface element. If the target computer has multiple Ethernet cards, you must select one of the cards for host-target communication. This option returns the index number of the card selected on the target computer upon starting.
LegacyMultiCoreConfig	MATLAB property values are 'on' and 'off'. The default value is 'off'. There is no corresponding Simulink Real-Time Explorer interface element. Set this value to 'on' only if your target computer contains hardware not compliant with the Advanced Configuration and Power Interface (ACPI) standard. Otherwise, set this value to 'off'.

Environment Property	Description
`MaxModelSize`	Supported MATLAB property values are `'1MB'` and `'4MB'`. The default value is `'1MB'`. Value `'16MB'` is not supported. From 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. Choosing the maximum model size reserves the specified amount of memory on the target computer for the target application. Memory not used by the target application is used by the kernel and by the heap for data logging. Selecting too high a value leaves less memory for data logging. Selecting too low a value does not reserve enough memory for the target application and creates an error. You can approximate the size of the target application by the size of the DLM file produced by the build process.
`MulticoreSupport`	MATLAB property values are `'on'` and `'off'`. The default value is `'off'`. If your target computer has multicore processors, select the Simulink Real-Time Explorer **Multicore CPU** check box to take advantage of these processors for background tasks. Otherwise, clear it.
`Name`	MATLAB property is the target computer name string. 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.

Environment Property	Description
`NonPentiumSupport`	MATLAB property values are `'on'` and `'off'`. The default value is `'off'`. 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. If your target computer has a Pentium or higher compatible processor, selecting this check box slows the performance of your target computer.
`SecondaryIDE`	MATLAB property values are `'on'` and `'off'`. The default value is `'off'`. If you want to use the disks connected to a secondary IDE controller, select the Simulink Real-Time Explorer **Secondary IDE** check box. Otherwise, clear it.
`ShowHardware`	MATLAB property values are `'on'` and `'off'`. The default value is `'off'`. There is no corresponding Simulink Real-Time Explorer interface element. 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 host computer cannot communicate with the target computer after the kernel starts with `ShowHardware` set.

Environment Property	Description
`TargetRAMSizeMB`	MATLAB property values are `'Auto'` and `'xxx'`. *`xxx`* is a positive value specifying the total amount of RAM, in megabytes, installed on the target computer. Target computer RAM is used for the kernel, target application, data logging, and other functions that use the heap. The default value is `'Auto'`. To allow the target application to read the target computer BIOS and determine the amount of memory up to a maximum of 2 GB, in Simulink Real-Time Explorer, click **RAM size Auto**. If the target 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 2 GB of memory.
`TargetScope`	MATLAB property values are `'Disabled'` and `'Enabled'`. The default value is `'Enabled'`. To display scope information graphically, set the Simulink Real-Time Explorer **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.
`USBSupport`	MATLAB property values are `'on'` and `'off'`. The default value is `'on'`. To use a USB port on the target computer, for example to connect a USB mouse, select the Simulink Real-Time Explorer **USB Support** check box. Otherwise, clear it.

Boot Configuration

Environment Property	Description
`BootFloppyLocation`	Drive name for creation of target boot disk.
`DOSLoaderLocation`	Location of DOS Loader files to start target computers from devices other than floppy disk or CD.
`TargetBoot`	MATLAB property values are `'BootFloppy'`, `'CDBoot'`, `'DOSLoader'`, `'NetworkBoot'`, and `'StandAlone'`. To specify a boot mode, from the Simulink Real-Time Explorer **Boot mode** list, select one of `Removable Disk`, `CD`, `DOS Loader`, `Network`, or `Stand Alone`. To create a bootable image for the specified boot mode, click **Create boot disk**.
`TargetMACAddress`	MATLAB property value is the physical target computer MAC address from which to accept start requests when starting within a dedicated network. Format the MAC address as six pairs of hexadecimal numbers, separated by colons: `xx:xx:xx:xx:xx:xx`

Environment Property	Description
	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.

See Also

`SimulinkRealTime.targetSettings.get`

SimulinkRealTime.targetSettings.setAsDefaultTarget

Purpose

Set specific target computer environment object as default

Syntax

`env_object.setAsDefaultTarget`

Description

Method of `SimulinkRealTime.target` objects. `makeDefault` sets the specified target computer environment object as the default target computer from the `SimulinkRealTime.target` class.

Examples

Set the specified target as the default target computer.

```
tgs=SimulinkRealTime.getTargetSettings('TargetPC1');
tgs.setAsDefaultTarget
```

See Also

`SimulinkRealTime.targetSettings.set` | `SimulinkRealTime.targetSettings.get`

SimulinkRealTime.fileSystem

Purpose

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.fileSystem (constructor)`	Create file system object

Methods

These methods are specific to class `SimulinkRealTime.fileSystem`.

Method	Description
`SimulinkRealTime.fileSystem.cd`	Change folder on target computer
`SimulinkRealTime.fileSystem.dir`	List contents of current folder on target computer
`SimulinkRealTime.fileSystem.diskinfo`	Information about target computer drive
`SimulinkRealTime.fileSystem.fclose`	Close open target computer file or files
`SimulinkRealTime.fileSystem.fileinfo`	Target computer file information
`SimulinkRealTime.fileSystem.filetable`	Information about open files in target computer file system
`SimulinkRealTime.fileSystem.fopen`	Open target computer file for reading
`SimulinkRealTime.fileSystem.fread`	Read open target computer file
`SimulinkRealTime.fileSystem.fwrite`	Write binary data to open target computer file
`SimulinkRealTime.fileSystem.getfilesize`	Size of file on target computer
`SimulinkRealTime.fileSystem.mkdir`	Make folder on target computer
`SimulinkRealTime.fileSystem.pwd`	Current folder path of target computer

Method	Description
`SimulinkRealTime.fileSystem.removefile`	Remove file from target computer
`SimulinkRealTime.fileSystem.rmdir`	Remove folder from target computer

SimulinkRealTime.fileSystem (constructor)

Purpose

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 file system object (`SimulinkRealTime.fileSystem`). The file system object represents the file system on the target computer. You work with the file system by changing the file system object using methods.

If you have one target computer, or if you designate a target computer as the default one in your system, use `filesys_object = SimulinkRealTime.fileSystem` to create a file system object.

If you have a target computer object in the Simulink Real-Time Explorer, use `filesys_object = SimulinkRealTime.fileSystem(target_object)` to construct a corresponding file system object from the MATLAB Command Window.

Examples

In the following example, a file system object for the default target computer is created.

```
fs1 = SimulinkRealTime.fileSystem
```

If you have an `SimulinkRealTime.target` object, you can construct an `SimulinkRealTime.fileSystem` object by passing the `SimulinkRealTime.target` object variable to the `SimulinkRealTime.fileSystem` constructor as an argument.

```
tg1 = SimulinkRealTime.target('TargetPC1');
fs2 = SimulinkRealTime.fileSystem(tg1)
```

Purpose

Change folder on target computer

Syntax

```
cd(file_obj,target_PC_dir)
file_obj.cd(target_PC_dir)
```

Arguments

`file_obj`	Name of the `SimulinkRealTime.fileSystem` object.
`target_PC_dir`	Name of the target computer folder to change.

Description

Method of `SimulinkRealTime.fileSystem` objects. From the host computer, changes the folder on the target computer.

Examples

For the file system object `fsys`, change the folder from the current one to one named `logs`.

```
cd(fsys,logs) or fsys.cd(logs)
```

See Also

`cd` | `SimulinkRealTime.fileSystem.mkdir` | `SimulinkRealTime.fileSystem.pwd`

SimulinkRealTime.fileSystem.dir

Purpose

List contents of current folder on target computer

Syntax

```
dir(file_obj)
```

Arguments

`file_obj`	Name of the `SimulinkRealTime.fileSystem` object.

Description

Method of `SimulinkRealTime.fileSystem` objects. From the host computer, lists the contents of the folder on the target computer.

To get the results in an M-by-1 structure, use a syntax like `ans=dir(file_obj)`. This syntax returns a structure like the following:

```
ans =
1x5 struct array with fields:
name
date
time
bytes
isdir
```

- `name` — Name of an object in the folder, shown as a cell array. The name, stored in the first element of the cell array, can have up to eight characters. The three-character file extension is stored in the second element of the cell array.
- `date` — The last date at which the object was saved.
- `time` — The last time at which the object was saved.
- `bytes` — Size in bytes of that object.
- `isdir` — If `1`, the object is a folder. If `0`, it is not a folder.

Examples

List the contents of the folder for the file system object `fsys`.

```
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.

See Also

`dir` | `SimulinkRealTime.fileSystem.mkdir` | `SimulinkRealTime.fileSystem.cd` | `SimulinkRealTime.fileSystem.pwd`

SimulinkRealTime.fileSystem.diskinfo

Purpose

Target computer drive configuration information

Syntax

```
filesys_obj.diskinfo(target_PC_drive)
diskinfo(filesys_obj,target_PC_drive)
```

Arguments

`filesys_obj`	Name of the `SimulinkRealTime.fileSystem` file system object.
`target_PC_drive`	Name of the target computer drive being accessed.

Description

`filesys_obj.diskinfo(target_PC_drive)` is called from the host computer and returns configuration information for the specified drive on the target computer. An alternative syntax is:

```
diskinfo(filesys_obj,target_PC_drive)
```

Examples

For file system object `fsys`, return configuration information for the target computer C:\ drive.

```
diskinfo(fsys,'C:\') or fsys.diskinfo('C:\')
ans =
                  Label: 'SYSTEM '
            DriveLetter: 'C'
               Reserved: ''
           SerialNumber: 1.0294e+009
    FirstPhysicalSector: 63
                FATType: 32
               FATCount: 2
          MaxDirEntries: 0
         BytesPerSector: 512
      SectorsPerCluster: 4
          TotalClusters: 2040293
            BadClusters: 0
           FreeClusters: 1007937
                  Files: 19968
             FileChains: 22480
             FreeChains: 1300
       LargestFreeChain: 64349
```

SimulinkRealTime.fileSystem.fclose

Purpose

Close target computer file

Syntax

```
fclose(filesys_obj,file_ID)
filesys_obj.fclose(file_ID)
```

Arguments

`filesys_obj`	Name of the `SimulinkRealTime.fileSystem` file system object.
`file_ID`	File identifier of the file to close.

Description

Method of `SimulinkRealTime.fileSystem` objects. From the host computer, closes one or more open files in the target computer file system (except standard input, output, and error). The `file_ID` argument is the file identifier associated with an open file. 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`.

```
fclose(fsys,h) or fsys.fclose(h)
```

See Also

`fclose` | `SimulinkRealTime.fileSystem.fopen` | `SimulinkRealTime.fileSystem.fread` | `SimulinkRealTime.fileSystem.filetable` | `SimulinkRealTime.fileSystem.fwrite`

Purpose

Target computer file configuration information

Syntax

```
fileinfo(filesys_obj,file_ID)
filesys_obj.fileinfo(file_ID)
```

Arguments

`filesys_obj`	Name of the `SimulinkRealTime.fileSystem` file system object.
`file_ID`	File identifier of the file for which to get file configuration information.

Description

From the host computer, 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`.

```
fileinfo(fsys,h) or fsys.fileinfo(h)
ans =
               FilePos: 0
         AllocatedSize: 12288
         ClusterChains: 1
    VolumeSerialNumber: 1.0450e+009
              FullName: 'C:\DATA.DAT'
```

SimulinkRealTime.fileSystem.filetable

Purpose

Information about open files in target computer file system

Syntax

```
filetable(filesys_obj)
filesys_obj.filetable
```

Arguments

`filesys_obj`	Name of the `SimulinkRealTime.fileSystem` file system object.

Description

Method of `SimulinkRealTime.fileSystem` objects. From the host computer, displays a table of the open files in the target computer file system. You cannot have more than eight files open at the same time in the file system.

Examples

Return a table of the open files in the target computer file system for the file system object `fsys`.

```
filetable(fsys) or fsys.filetable
ans =
Index    Handle  Flags     FilePos  Name
-----------------------------------------
    0  00060000  R__          8512  C:\DATA.DAT
    1  00080001  R__             0  C:\DATA1.DAT
    2  000A0002  R__          8512  C:\DATA2.DAT
    3  000C0003  R__          8512  C:\DATA3.DAT
    4  001E000S  R__             0  C:\DATA4.DAT
```

The table returns the open file handles in hexadecimal. To convert a 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`.

```
fsys.fclose(h1);
```

See Also

SimulinkRealTime.fileSystem.fopen | SimulinkRealTime.fileSystem.fclose | hex2dec

SimulinkRealTime.fileSystem.fopen

Purpose

Open target computer file for reading

Syntax

```
file_ID = fopen(file_obj,'file_name')
file_ID = file_obj.fopen('file_name')
file_ID = fopen(file_obj,'file_name',permission)
file_ID = file_obj.fopen('file_name',permission)
```

Arguments

`file_obj`	Name of the `SimulinkRealTime.fileSystem` object.
`'file_name'`	Name of the target computer to open.
`permission`	Values are `'r'`, `'w'`, `'a'`, `'r+'`, `'w+'`, or `'a+'`. This argument is optional with `'r'` as the default value.

Description

Method of `SimulinkRealTime.fileSystem` objects. From the host computer, opens the specified file name on the target computer for binary access.

The permission argument values are

- `'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.

You cannot have more than eight files open at one 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`. Assign the resulting file handle to a variable for reading.

```
h = fopen(fsys,'data.dat') or fsys.fopen('data.dat')
ans =
     2883584
d = fread(fsys,h);
```

See Also

`fopen` | `SimulinkRealTime.fileSystem.fclose` | `SimulinkRealTime.fileSystem.fread` | `SimulinkRealTime.fileSystem.fwrite`

SimulinkRealTime.fileSystem.fread

Purpose

Read open target computer file

Syntax

```
A = file_obj.fread(file_ID)
A = fread(file_obj,file_ID)
A = file_obj.fread(file_ID,offset,numbytes)
A = fread(file_obj,file_ID,offset,numbytes)
```

Arguments

`file_obj`	Name of the `SimulinkRealTime.fileSystem` object.
`file_ID`	File identifier of the file to read.
`offset`	Position from the beginning of the file from which `fread` can start to read.
`numbytes`	Maximum number of bytes `fread` can read.

Description

`A = file_obj.fread(file_ID)` reads binary data from the file on the target computer and writes it into matrix `A`. The `file_ID` argument is the file identifier associated with an open file. An alternative syntax is:

`A = fread(file_obj,file_ID)`

`A = file_obj.fread(file_ID,offset,numbytes)` reads a block of bytes from `file_ID` and writes the block into matrix `A`. An alternative syntax is:

`A = fread(file_obj,file_ID,offset,numbytes)`

The `offset` argument specifies the position from the beginning of the file from which this function can start to read. `numbytes` specifies the maximum number of bytes to read.

To get a count of the total number of bytes read into `A`, use the following:

```
count = length(A);
```

`length(A)` might be less than the number of bytes requested if that number of bytes are not currently available. It is zero if the operation reaches the end of the file.

This is a method of `SimulinkRealTime.fileSystem` objects called from the host computer.

Examples

Open the file `data.dat` in the target computer file system object `fsys`. Assign the resulting file handle to a variable for reading.

```
h=fsys.fopen('data.dat')
d=fread(fsys,h);
```

This 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`

SimulinkRealTime.fileSystem.fwrite

Purpose

Write binary data to open target computer file

Syntax

```
fwrite(file_obj,file_ID,A)
file_obj.fwrite(file_ID,A)
```

Arguments

`file_obj`	Name of the `SimulinkRealTime.fileSystem` object.
`file_ID`	File identifier of the file to write.
`A`	Elements of matrix `A` to be written to the specified file.

Description

Method of `SimulinkRealTime.fileSystem` objects. From the host computer, writes the elements of matrix `A` to the file identified by `file_ID`. The data is written to the file in column order. The `file_ID` argument is the file identifier associated with an open file. `fwrite` requires that the file be open with write permission.

Examples

Open the file `data.dat` in the target computer file system object `fsys`. Assign the resulting file handle to a variable for writing.

```
h = fopen(fsys,'data.dat','w')
```

or

```
fsys.fopen('data.dat','w')

ans =
     2883584
d = fwrite(fsys,h,magic(5));
```

This writes the elements of matrix `A` to the file handle h. This content is written in column order.

See Also

`fwrite` | `SimulinkRealTime.fileSystem.fclose` | `SimulinkRealTime.fileSystem.fopen` | `SimulinkRealTime.fileSystem.fread`

Purpose

Size of file on target computer

Syntax

```
getfilesize(file_obj,file_ID)
file_obj.getfilesize(file_ID)
```

Arguments

`file_obj`	Name of the `SimulinkRealTime.fileSystem` object.
`file_ID`	File identifier of the file to get the size of.

Description

Method of `SimulinkRealTime.fileSystem` objects. From the host computer, gets the size (in bytes) of the file identified by the `file_ID` file identifier on the target computer file system. Use the 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`.

```
getfilesize(fsys,h) or fsys.getfilesize(h)
```

See Also

`SimulinkRealTime.fileSystem.fopen`

SimulinkRealTime.fileSystem.mkdir

Purpose

Make folder on target computer

Syntax

```
mkdir(file_obj,dir_name)
file_obj.mkdir(dir_name)
```

Arguments

`file_obj`	Name of the `SimulinkRealTime.fileSystem` object.
`dir_name`	Name of the folder to be created.

Description

Method of `SimulinkRealTime.fileSystem` objects. From the host computer, makes a new folder in the current folder on the target computer file system.

Note that to delete a folder from the target computer, you must restart the computer into DOS or some other operating system and use a utility in that system to delete the folder.

Examples

Create a new folder, `logs`, in the target computer file system object `fsys`.

```
mkdir(fsys,logs)
```

or

```
fsys.mkdir(logs)
```

See Also

`mkdir` | `SimulinkRealTime.fileSystem.dir` | `SimulinkRealTime.fileSystem.pwd`

SimulinkRealTime.fileSystem.pwd

Purpose

Current folder path of target computer

Syntax

```
pwd(file_obj)
file_obj.pwd
```

Arguments

`file_obj`	Name of the `SimulinkRealTime.fileSystem` object.

Description

Method of `SimulinkRealTime.fileSystem` object. Returns the pathname of the current target computer folder.

Examples

Return the target computer current folder for the file system object `fsys`.

```
pwd(fsys) or fsys.pwd
```

See Also

`pwd` | `SimulinkRealTime.fileSystem.dir` | `SimulinkRealTime.fileSystem.mkdir`

SimulinkRealTime.fileSystem.removefile

Purpose

Remove file from target computer

Syntax

```
removefile(file_obj,file_name)
file_obj.removefile(file_name)
```

Arguments

`file_name`	Name of the file to remove from the target computer file system.
`file_obj`	Name of the `SimulinkRealTime.fileSystem` object.

Description

Method of `SimulinkRealTime.fileSystem` objects. Removes a file from the target computer file system.

Note You cannot recover this file once it is removed.

Examples

Remove the file `data2.dat` from the target computer file system `fsys`.

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

or

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

SimulinkRealTime.fileSystem.rmdir

Purpose

Remove folder from target computer

Syntax

```
rmdir(file_obj,dir_name)
file_obj.rmdir(dir_name)
```

Arguments

`dir_name`	Name of the folder to remove from the target computer file system.
`file_obj`	Name of the `SimulinkRealTime.fileSystem` object.

Description

Method of `SimulinkRealTime.fileSystem` object. Removes a folder from the target computer file system.

Note You cannot recover this folder once it is removed.

Examples

Remove the folder `data2dir.dat` from the target computer file system `fsys`.

```
rmdir(f,'data2dir.dat')
```

or

```
fsys.rmdir('data2dir.dat')
```

SimulinkRealTime.fileSystem.selectdrive

Purpose

Select target computer drive

Syntax

```
selectdrive(file_obj,'drive')
file_obj.selectdrive('drive')
```

Arguments

`drive`	Name of the drive to set.
`file_obj`	Name of the `SimulinkRealTime.fileSystem` object.

Description

Method of `SimulinkRealTime.fileSystem` objects. `selectdrive` sets the current drive of the target computer to the specified string. Enter the drive string with an extra backslash (\). For example, `D:\\` for the D:\ drive.

Note Use the `SimulinkRealTime.fileSystem.cd` method instead to get the same behavior.

Examples

Set the current target computer drive to D:\.

```
selectdrive(fsys,'D:\\')
```

or

```
fsys.selectdrive('D:\\')
```

Purpose

Target object representing target application

Description

Provides access to methods and properties used to start and stop the target application, read and set parameters, monitor signals, and retrieve status information about the target computer.

Constructor

Constructor	Description
`SimulinkRealTime.target (constructor)`	Create target object representing target application

Methods

Method	Description
`SimulinkRealTime.target.addscope`	Create scopes
`SimulinkRealTime.target.close`	Close serial port connecting host computer with target computer
`SimulinkRealTime.target.get`	Return target application object property values
`SimulinkRealTime.target.getlog`	All or part of output logs from target object
`SimulinkRealTime.target.getparam`	Value of target object parameter index
`SimulinkRealTime.target.getparamid`	Parameter index from parameter list
`SimulinkRealTime.target.getparamname`	Block path and parameter name from index list
`SimulinkRealTime.target.getscope`	Scope object pointing to scope defined in kernel
`SimulinkRealTime.target.getsignal`	Value of target object signal index
`SimulinkRealTime.target.getsignalid`	Signal index or signal property from signal list
`SimulinkRealTime.target.getsignalidsfromlabel`	Return vector of signal indices
`SimulinkRealTime.target.getsignallabel`	Return signal label
`SimulinkRealTime.target.getsignalname`	Signal name from index list
`SimulinkRealTime.target.load`	Download target application to target computer

Method	Description
`SimulinkRealTime.target.loadparamset`	Restore parameter values saved in specified file
`SimulinkRealTime.target.ping`	Tests communication between host and target computers
`SimulinkRealTime.target.reboot`	Reboot target computer
`SimulinkRealTime.target.remscope`	Remove scope from target computer
`SimulinkRealTime.target.saveparamset`	Save current target application parameter values
`SimulinkRealTime.target.set`	Change target application object property values
`SimulinkRealTime.target.setparam`	Change writable target object parameters
`SimulinkRealTime.target.start`	Start execution of target application on target computer
`SimulinkRealTime.target.stop`	Stop execution of target application on target computer
`SimulinkRealTime.target.unload`	Remove current target application from target computer

Properties

Properties are read using `SimulinkRealTime.target.get`. Writable properties are written using `SimulinkRealTime.target.set`.

Property	Description	Writable
`Application`	Name of the Simulink model and target application built from that model.	No
`AvgTET`	Average task execution time. This value is an average of the measured CPU times, in seconds, to run the model equations and post outputs during each sample interval. Task execution time is nearly constant, with minor deviations due to cache, memory access, interrupt latency, and multirate model execution. The TET includes: • Complete I/O latency.	No

Property	Description	Writable
	• Data logging (the parts that happen in a real-time task). This includes data captured in scopes. • Asynchronous interruptions. • Parameter updating latency (if the **Double buffer parameter changes** parameter is set in the **Simulink Real-Time Options** node of the model Configuration Parameters dialog box). Note that the TET is not the only consideration in determining the minimum achievable sample time. Other considerations, not included in the TET, are: • Time required to measure TET • Interrupt latency required to schedule and run one step of the model	
`CommunicationTimeOut`	Communication timeout between host and target computer, in seconds.	Yes
`Connected`	Communication status between the host computer and the target computer. Values are `'Yes'` and `'No'`.	No
`CPUoverload`	CPU status for overload. If the target application requires more CPU time than the sample time of the model, this value is set from `'none'` to `'detected'` and the current run is stopped. Returning this status to `'none'` requires either a faster processor or a larger sample time.	No

Property	Description	Writable
`ExecTime`	Execution time. Time, in seconds, since your target application started running. When the target application stops, the total execution time is displayed.	No
`LogMode`	Controls which data points are logged: • Time-equidistant logging. Logs a data point at every time interval. Set value to `'Normal'`. • Value-equidistant logging. Logs a data point only when an output signal from the `OutputLog` changes by a specified value (increment). Set the value to the difference in signal values.	Yes
`MaxLogSamples`	Maximum number of samples for each logged signal within the circular buffers for `TimeLog`, `StateLog`, `OutputLog`, and `TETLog`. `StateLog` and `OutputLog` can have one or more signals. This value is calculated by dividing the **Signal Logging Buffer Size** by the number of logged signals. The **Signal Logging Buffer Size** box is in the **Simulink Real-Time Options** pane of the Configuration Parameters dialog box.	No
`MaxTET`	Maximum task execution time. Corresponds to the slowest time (longest time measured), in seconds, to update model equations and post outputs.	No

Property	Description	Writable
`MinTET`	Minimum task execution time. Corresponds to the fastest time (smallest time measured), in seconds, to update model equations and post outputs.	No
`Mode`	Type of Simulink Coder code generation. Values are `'Real-Time Singletasking'`, `'Real-Time Multitasking'`, and `'Accelerate'`. The default value is `'Real-Time Singletasking'`. Even if you select `'Real-Time Multitasking'`, the actual mode can be `'Real-Time Singletasking'`. This happens if your model contains only one or two tasks and the sample rates are equal.	No
`NumLogWraps`	The number of times the circular buffer wrapped. The buffer wraps each time the number of samples exceeds `MaxLogSamples`.	No
`NumParameters`	The number of parameters from your Simulink model that you can tune or change.	No
`NumSignals`	The number of signals from your Simulink model that are available to be viewed with a scope.	No
`OutputLog`	Storage in the MATLAB workspace for the output or Y-vector logged during execution of the target application.	No

Property	Description	Writable
`Parameters`	List of tunable parameters. This list is visible only when `ShowParameters` is set to `'on'`: • Property value. Value of the parameter in a Simulink block. • Type. Data type of the parameter. Always `double`. • Size. Size of the parameter. For example, scalar, 1-by-2 vector, or 2-by-3 matrix. • Parameter name. Name of a parameter in a Simulink block. • Block name. Name of a Simulink block.	No
`SampleTime`	Time between samples. This value equals the step size, in seconds, for updating the model equations and posting the outputs. (See “Alternative Configuration and Control Methods” for limitations on target property changes to sample times.)	Yes
`Scopes`	List of index numbers, with one index for each scope.	No
`SessionTime`	Time since the kernel started running on your target computer. This is also the elapsed time since you started the target computer. Values are in seconds.	No
`ShowParameters`	Flag set to view or hide the list of parameters from your Simulink blocks. This list is shown when you display the properties for a target object. Values are `'on'` and `'off'`.	Yes

Property	Description	Writable
`ShowSignals`	Flag set to view or hide the list of signals from your Simulink blocks. This list is shown when you display the properties for a target object. Values are `'on'` and `'off'`.	Yes
`Signals`	List of viewable signals. This list is visible only when `ShowSignals` is set to `'on'`. • Property name. `S0`, `S1`. . . • Property value. Value of the signal. • Block name. Name of the Simulink block the signal is from.	No
`StateLog`	Storage in the MATLAB workspace for the state or x-vector logged during execution of the target application.	No
`Status`	Execution status of your target application. Values are `'stopped'` and `'running'`.	No
`StopTime`	Time when the target application stops running. Values are in seconds. The original value is set in the **Solver** pane of the Configuration Parameters dialog box. When the `ExecTime` reaches `StopTime`, the application stops running.	Yes

Property	Description	Writable
`TETLog`	Storage in the MATLAB workspace for a vector containing task execution times during execution of the target application. To enable logging of the TET, you must select the **Log Task Execution Time** check box in the **Simulink Real-Time Options** pane of the Configuration Parameters dialog box.	No
`TimeLog`	Storage in the MATLAB workspace for the time or T-vector logged during execution of the target application.	No
`ViewMode`	Display either all scopes or a single scope on the target computer. Value is `'all'` or a single scope index. This property is active only if the environment property `TargetScope` is set to `enabled`.	Yes

SimulinkRealTime.target (constructor)

Purpose

Create object to manage 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.

`target_object = SimulinkRealTime.target(target_name)` constructs a target object representing the target computer designated by `target_name`.

Input Arguments

target_name - Name assigned to target computer
string

Example: 'TargetPC1'

Data Types
`char`

Output Arguments

target_object - Target object representing target computer
structure

Examples

Default target computer

Creates a target object to communicate with the default target computer, assumed to be connected.

```
target_object = SimulinkRealTime.target

Target: TargetPC1
   Connected            = Yes
   Application          = loader
```

Specific target computer

Creates a target object to communicate with target computer `TargetPC1`, assumed to be not connected.

```
target_object = SimulinkRealTime.target('TargetPC1')

Target: TargetPC1
   Connected            = No
```

See Also

slrt | SimulinkRealTime.TargetSettings |
SimulinkRealTime.target.get | SimulinkRealTime.target.set

Purpose

Create scopes

Syntax

Create a scope and scope object without assigning to a MATLAB variable.

```
addscope(target_object, scope_type, scope_number)
target_object.addscope(scope_type, scope_number)
```

Create a scope, scope object, and assign to a MATLAB variable

```
scope_object = addscope(target_object,
    scope_type, scope_number)
scope_object = target_object.addscope(scope_type,
    scope_number)
```

Target computer command line — When you are using this command on the target computer, you can only add a target scope.

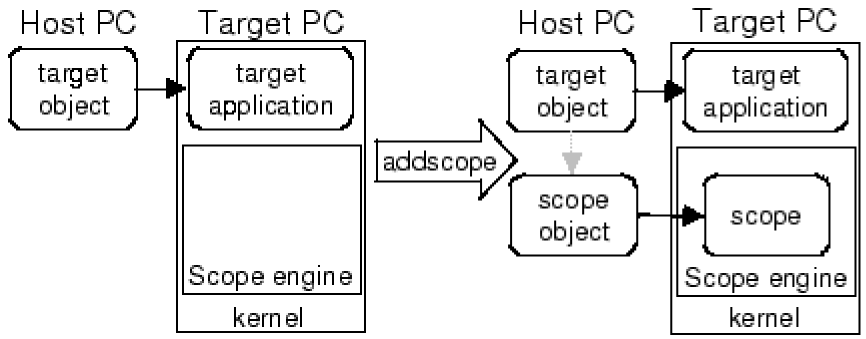
```
addscope
addscope scope_number
```

Arguments

`target_object`	Name of a target object. The default target name is `tg`.
`scope_type`	Values are `'host'`, `'target'`, or `'file'`. This argument is optional with `host` as the default value.
`scope_number`	Vector of new scope indices. This argument is optional. The next available integer in the target object property `Scopes` as the default value. If you enter a scope index for an existing scope object, the result is an error.

Description

`addscope` creates a scope of the specified type and updates the target object property `Scopes`. This method returns a scope object vector. If the result is not assigned to a variable, the scope object properties are listed in the MATLAB window. The Simulink Real-Time product supports 10 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.



Examples

Create a scope and scope object `sc1` using the method `addscope`. A target scope is created on the target computer with an index of `1`, and a scope object is created on the host computer, assigned to the variable `sc1`. The target object property `Scopes` is changed from `No scopes defined` to `1`.

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

or

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

Create a scope with the method `addscope` and then create a scope object, corresponding to this scope, using the method `getscope`. A target scope is created on the target computer with an index of `1`, and a scope object is created on the host computer, but it is not assigned to a variable. The target object property `Scopes` is changed from `No scopes defined` to `1`.

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

Create two scopes using a vector of scope objects `scvector`. Two target scopes are created on the target computer with scope indices of `1` and `2`, and two scope objects are created on the host computer that represent the scopes on the target computer. The target object property `Scopes` is changed from `No scopes defined` to `1,2`.

```
scvector = addscope(tg, 'target', [1, 2])
```

Create a scope and scope object `sc4` of type `file` using the method `addscope`. A file scope is created on the target computer with an index of `4`. A scope object is created on the host computer and is assigned to the variable `sc4`. The target object property `Scopes` is changed from `No scopes defined` to `4`.

```
sc4 = addscope(tg,'file',4) or sc4 = tg.addscope('file',4)
```

See Also

`SimulinkRealTime.target.remscope` |
`SimulinkRealTime.target.getscope`

How To

- "Target Scope Usage"
- "Host Scope Usage"
- "File Scope Usage"
- "Application and Driver Scripts"

SimulinkRealTime.target.close

Purpose

Close serial port connecting host computer with target computer

Syntax

```
close(target_object)
target_object.close
```

Arguments

`target_object`	Name of a target object.

Description

`close` closes the serial connection between the host computer and a target computer. If you want to use the serial port for another function without quitting the MATLAB window – for example, a modem – use this function to close the connection.

Purpose

Return target application object property values

Syntax

```
get(target_object, 'target_object_property')
```

Arguments

`target_object`	Name of a target object.
`'target_object_property'`	Name of a target object property.

Description

`get` gets the value of readable target object properties from a target object.

The properties for a target object are listed in the following table. This table includes a description of the properties and which properties you can change directly by assigning a value.

Property	Description	Writable
`Application`	Name of the Simulink model and target application built from that model.	No
`AvgTET`	Average task execution time. This value is an average of the measured CPU times, in seconds, to run the model equations and post outputs during each sample interval. Task execution time is nearly constant, with minor deviations due to cache, memory access, interrupt latency, and multirate model execution. The TET includes: • Complete I/O latency. • Data logging (the parts that happen in a real-time task). This includes data captured in scopes. • Asynchronous interruptions.	No

Property	Description	Writable
	• Parameter updating latency (if the **Double buffer parameter changes** parameter is set in the **Simulink Real-Time Options** node of the model Configuration Parameters dialog box). Note that the TET is not the only consideration in determining the minimum achievable sample time. Other considerations, not included in the TET, are: • Time required to measure TET • Interrupt latency required to schedule and run one step of the model	
`CommunicationTimeOut`	Communication timeout between host and target computer, in seconds.	Yes
`Connected`	Communication status between the host computer and the target computer. Values are `'Yes'` and `'No'`.	No
`CPUoverload`	CPU status for overload. If the target application requires more CPU time than the sample time of the model, this value is set from `'none'` to `'detected'` and the current run is stopped. Returning this status to `'none'` requires either a faster processor or a larger sample time.	No
`ExecTime`	Execution time. Time, in seconds, since your target application started running. When the target application stops, the total execution time is displayed.	No

Property	Description	Writable
`LogMode`	Controls which data points are logged: • Time-equidistant logging. Logs a data point at every time interval. Set value to `'Normal'`. • Value-equidistant logging. Logs a data point only when an output signal from the `OutputLog` changes by a specified value (increment). Set the value to the difference in signal values.	Yes
`MaxLogSamples`	Maximum number of samples for each logged signal within the circular buffers for `TimeLog`, `StateLog`, `OutputLog`, and `TETLog`. `StateLog` and `OutputLog` can have one or more signals. This value is calculated by dividing the **Signal Logging Buffer Size** by the number of logged signals. The **Signal Logging Buffer Size** box is in the **Simulink Real-Time Options** pane of the Configuration Parameters dialog box.	No
`MaxTET`	Maximum task execution time. Corresponds to the slowest time (longest time measured), in seconds, to update model equations and post outputs.	No
`MinTET`	Minimum task execution time. Corresponds to the fastest time (smallest time measured), in seconds, to update model equations and post outputs.	No

Property	Description	Writable
`Mode`	Type of Simulink Coder code generation. Values are `'Real-Time Singletasking'`, `'Real-Time Multitasking'`, and `'Accelerate'`. The default value is `'Real-Time Singletasking'`. Even if you select `'Real-Time Multitasking'`, the actual mode can be `'Real-Time Singletasking'`. This happens if your model contains only one or two tasks and the sample rates are equal.	No
`NumLogWraps`	The number of times the circular buffer wrapped. The buffer wraps each time the number of samples exceeds `MaxLogSamples`.	No
`NumParameters`	The number of parameters from your Simulink model that you can tune or change.	No
`NumSignals`	The number of signals from your Simulink model that are available to be viewed with a scope.	No
`OutputLog`	Storage in the MATLAB workspace for the output or Y-vector logged during execution of the target application.	No

Property	Description	Writable
`Parameters`	List of tunable parameters. This list is visible only when `ShowParameters` is set to `'on'`: • Property value. Value of the parameter in a Simulink block. • Type. Data type of the parameter. Always `double`. • Size. Size of the parameter. For example, scalar, 1-by-2 vector, or 2-by-3 matrix. • Parameter name. Name of a parameter in a Simulink block. • Block name. Name of a Simulink block.	No
`SampleTime`	Time between samples. This value equals the step size, in seconds, for updating the model equations and posting the outputs. (See "Alternative Configuration and Control Methods" for limitations on target property changes to sample times.)	Yes
`Scopes`	List of index numbers, with one index for each scope.	No
`SessionTime`	Time since the kernel started running on your target computer. This is also the elapsed time since you started the target computer. Values are in seconds.	No
`ShowParameters`	Flag set to view or hide the list of parameters from your Simulink blocks. This list is shown when you display the properties for a target object. Values are `'on'` and `'off'`.	Yes

Property	Description	Writable
`ShowSignals`	Flag set to view or hide the list of signals from your Simulink blocks. This list is shown when you display the properties for a target object. Values are `'on'` and `'off'`.	Yes
`Signals`	List of viewable signals. This list is visible only when `ShowSignals` is set to `'on'`. • Property name. `S0`, `S1`. . . • Property value. Value of the signal. • Block name. Name of the Simulink block the signal is from.	No
`StateLog`	Storage in the MATLAB workspace for the state or x-vector logged during execution of the target application.	No
`Status`	Execution status of your target application. Values are `'stopped'` and `'running'`.	No
`StopTime`	Time when the target application stops running. Values are in seconds. The original value is set in the **Solver** pane of the Configuration Parameters dialog box. When the `ExecTime` reaches `StopTime`, the application stops running.	Yes

Property	Description	Writable
`TETLog`	Storage in the MATLAB workspace for a vector containing task execution times during execution of the target application. To enable logging of the TET, you must select the **Log Task Execution Time** check box in the **Simulink Real-Time Options** pane of the Configuration Parameters dialog box.	No
`TimeLog`	Storage in the MATLAB workspace for the time or T-vector logged during execution of the target application.	No
`ViewMode`	Display either all scopes or a single scope on the target computer. Value is `'all'` or a single scope index. This property is active only if the environment property `TargetScope` is set to `enabled`.	Yes

Examples

List the value for the target object property `StopTime`. Notice that the property name is a string, in quotation marks, and not case sensitive.

```
get(tg,'stoptime') or tg.get('stoptime')
ans = 0.2
```

See Also

`SimulinkRealTime.target.set` | `get` | `SimulinkRealTime.fileScope.get` | `SimulinkRealTime.hostScope.get` | `SimulinkRealTime.targetScope.get`

SimulinkRealTime.target.getlog

Purpose

All or part of output logs from target object

Syntax

```
log = getlog(target_object, 'log_name', first_point,
number_samples, decimation)
```

Arguments

`log`	User-defined MATLAB variable.
`'log_name'`	Values are `TimeLog`, `StateLog`, `OutputLog`, or `TETLog`. This argument is required.
`first_point`	First data point. The logs begin with 1. This argument is optional. Default is 1.
`number_samples`	Number of samples after the start time. This argument is optional. Default is all points in log.
`decimation`	`1` returns all sample points. `n` returns every `n`th sample point. This argument is optional. Default is `1`.

Description

Use this function instead of the function `get` when you want only part of the data.

Examples

To get the first 1000 points in a log,

```
Out_log = getlog(tg, 'TETLog', 1, 1000)
```

To get every other point in the output log and plot values,

```
Output_log = getlog(tg, 'TETLog', 1, 10, 2)
Time_log = getlog(tg, 'TimeLog', 1, 10, 2)
plot(Time_log, Output_log)
```

See Also

`SimulinkRealTime.target.get`

How To

- "Set Configuration Parameters"

Purpose

Value of target object parameter index

Syntax

```
getparam(target_object, parameter_index)
```

Arguments

`target_object`	Name of a target object. The default name is `tg`.
`parameter_index`	Index number of the parameter.

Description

`getparam` returns the value of the parameter associated with `parameter_index`.

Examples

Get the value of parameter index `5`.

```
getparam(tg, 5)
ans = 400
```

SimulinkRealTime.target.getparamid

Purpose

Parameter index from parameter list

Syntax

```
getparamid(target_object, 'block_name', 'parameter_name')
```

Arguments

`target_object`	Name of a target object. The default name is `tg`.
`'block_name'`	Simulink block path without model name.
`'parameter_name'`	Name of a parameter within a Simulink block.

Description

`getparamid` returns the index of a parameter in the parameter list based on the path to the parameter name. The names must be entered in full and are case sensitive. Note, enter for `block_name` the mangled name that Simulink Coder uses for code generation.

Examples

Get the parameter property for the parameter `Gain` in the Simulink block `Gain1`, incrementally increase the gain, and pause to observe the signal trace.

```
id = getparamid(tg, 'Subsystem/Gain1', 'Gain')
for i = 1 : 3
  set(tg, id, i*2000);
  pause(1);
end
```

Get the property index of a single block.

```
getparamid(tg, 'Gain1', 'Gain') ans = 5
```

See Also

`SimulinkRealTime.target.getsignalid`

How To

- "Application and Driver Scripts"
- "Why Does the getparamid Function Return Nothing?"

SimulinkRealTime.target.getparamname

Purpose

Block path and parameter name from index list

Syntax

```
getparamname(target_object, parameter_index)
```

Arguments

`target_object`	Name of a target object. The default name is `tg`.
`parameter_index`	Index number of the parameter.

Description

`getparamname` returns two argument strings, block path and parameter name, from the index list for the specified parameter index.

Examples

Get the block path and parameter name of parameter index `5`.

```
[blockPath,parName]=getparamname(tg,5)
blockPath =
Signal Generator
parName =
Amplitude
```

SimulinkRealTime.target.getPCIInfo

Purpose

Determine PCI boards installed in target computer

Syntax

```
target_object.getPCIInfo
target_object.getPCIInfo('all')
target_object.getPCIInfo('verbose')

pci_devices = target_object.getPCIInfo( ___ )

target_object.getPCIInfo('supported')
pci_devices_supported =
target_object.getPCIInfo('supported')
```

Description

`target_object.getPCIInfo` without an argument 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 call displays in the Command Window information about the PCI devices 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 meet the following preconditions:

- The host-target communication link must be working. Before you can use `target_object.getPCIInfo`, the function `SimulinkRealTime.target.pingTarget` must return `success`.

- Either a target application is loaded or the loader is active. Before building the model, you can use `target_object.getPCIInfo` to find resources to enter into a driver block dialog box. Such resources include PCI bus number, slot number, and assigned IRQ number.

`target_object.getPCIInfo('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).

`target_object.getPCIInfo('verbose')` shows the information displayed by `target_object.getPCIInfo('all')` for the target computer represented by `target_object`, plus information about the PCI addresses assigned to this board by the BIOS.

`pci_devices = target_object.getPCIInfo( ___ )` queries the target computer represented by `target_object` according to the argument supplied and returns a structure containing information about the PCI devices found.

`target_object.getPCIInfo('supported')` displays a list of the PCI devices currently supported by the Simulink Real-Time block library. This call does not access the target computer, so host-target communication does not have to be active.

`pci_devices_supported = target_object.getPCIInfo('supported')` returns a structure containing a list of PCI devices currently supported by the Simulink Real-Time block library. This call does not access the target computer, so host-target communication does not have to be active.

Input Arguments

target_object - Object representing target computer

object created by `slrt`

Object representing the target computer being queried, as returned by `slrt`.

Example: `target_object = slrt('TargetPC1')`

Data Types
`function_handle`

Output Arguments

pci_devices - Information about the PCI devices in the target computer

vector

The vector returned by `getPCIInfo` without an argument contains information only for those PCI devices supported by Simulink Real-Time blocks. The vectors returned by `getPCIInfo` with the arguments `'all'` and `'verbose'` contain information about all PCI devices in the target computer and are identical.

The fields in this structure are:

Bus - PCI bus where device resides

scalar

`Bus` and `Slot` are used together to uniquely identify the location of a device or bus adapter in the target computer.

Slot - PCI slot where device resides

scalar

`Slot` and `Bus` are used together to 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 assigned by the PCI standards organization 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 assigned by the manufacturer to this device or bus adapter.

SubVendorID - Identifier for manufacturer of subsystem

string

Hexadecimal numeric string containing the identifier assigned by the PCI standards organization 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 assigned by the manufacturer 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` together 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` together identify the type and function of the device.

Interrupt - IRQ used by the device

scalar

Provides the board-level interrupt used by the device or bus adapter 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 by this device or bus adapter, the vector contains a structure with the following fields:

AddressSpaceIndicator - Indicates whether the address is a memory or I/O address

0 | 1

- `0` — Address is memory address
- `1` — Address is I/O address

BaseAddress - Memory address used by the device

string

Hexadecimal string containing the base memory address used by the device.

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 not prefetchable
- `1` — Address 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 device is supported by the Simulink Real-Time block library, contains the MATLAB and Simulink release version in which the driver was released. Otherwise, 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 represented by blocks in the Simulink Real-Time block library.

The fields are as follows:

VendorID - Identifier for manufacturer of the device
string

Hexadecimal numeric string containing the identifier assigned by the PCI standards organization 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 assigned by the manufacturer to this device or bus adapter.

SubVendorID - Identifier for manufacturer of subsystem
string

Hexadecimal numeric string containing the identifier assigned by the PCI standards organization 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 assigned by the manufacturer 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 device is supported by the Simulink Real-Time block library, contains the MATLAB and Simulink release version in which the driver was released. Otherwise, contains an empty vector.

Notes - Additional information about the device
string

Contains additional description of the device or bus adapter.

Examples

Display information for PCI devices that are supported by Simulink Real-Time block library on default computer

Start the default target computer with the Simulink Real-Time kernel. Verify the connection between the host and the target computer. At the MATLAB command prompt, type the command on the host computer.

```
slrtpingtarget

tg.getPCIInfo

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 all PCI devices on default computer

Start the default target computer with the Simulink Real-Time kernel. Verify the connection between the host and the target computer. At the MATLAB command prompt, type the command on the host computer.

```
slrtpingtarget

tg.getPCIInfo('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 host and the target computer. At the MATLAB command prompt, type the command on the host computer.

```
slrtpingtarget

tg.getPCIInfo('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 that are supported by Simulink Real-Time block library on default computer

Start the default target computer with the Simulink Real-Time kernel. Verify the connection between the host and the target computer. At the MATLAB command prompt, type the command on the host computer. Display the first structure in the vector.

```
slrtpingtarget

pci_devices = tg.getPCIInfo;
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 host and the target computer. At the MATLAB command prompt, type the command on the host computer. Display the first structure in the vector.

```
slrtpingtarget

pci_devices = tg.getPCIInfo('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. Get the `target_object` using `SimulinkRealTime.target`. Verify the connection between the host and the target computer. At the MATLAB prompt, type the command on the host computer. Display the first structure in the vector.

```
target_object = slrt('XPCLABTGT4');
target_object.pingTarget

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 all PCI devices supported by Simulink Real-Time block library

At the MATLAB prompt, type the commands on the host computer. The target computer need not be active.

```
target_object = SimulinkRealTime.target

target_object.getPCIInfo('supported')
```

```
List of supported PCI devices:

Vendor        Device            Type . . .

ADDI-DATA     APCI-1710         Inc. Encoder . . .
ADLINK        PCI-6208A         AO DI DO . . .
.
.
.
Speedgoat     IO321 (PMC-FPGA)  AI (IO321-5) . . .
Speedgoat     IO331 (PMC-FPGA)  DI DO (LVDS/LVCMOS) . . .
```

Return all PCI devices supported by Simulink Real-Time block library

At the MATLAB prompt, type the commands on the host computer. The target computer need not be active.

```
target_object = SimulinkRealTime.target

pci_devices_supported = target_object.getPCIInfo('supported');
pci_devices_supported(1)

ans =

       VendorID: '10e8'
       DeviceID: '818f'
    SubVendorID: '-1'
    SubDeviceID: '-1'
     DeviceName: 'APCI-1710'
     VendorName: 'ADDI-DATA'
     DeviceType: 'Inc. Encoder'
         DAChan: '0'
         ADChan: '0'
        DIOChan: '0'
        Release: 'R14SP2 or Earlier'
          Notes: ''
```

Related Examples

- “Where to Find PCI Board Information”
- “Command-Line Ethernet Card Selection by Index”

Concepts

- “PCI Bus I/O Devices”

Purpose

Scope object pointing to scope defined in kernel

Syntax

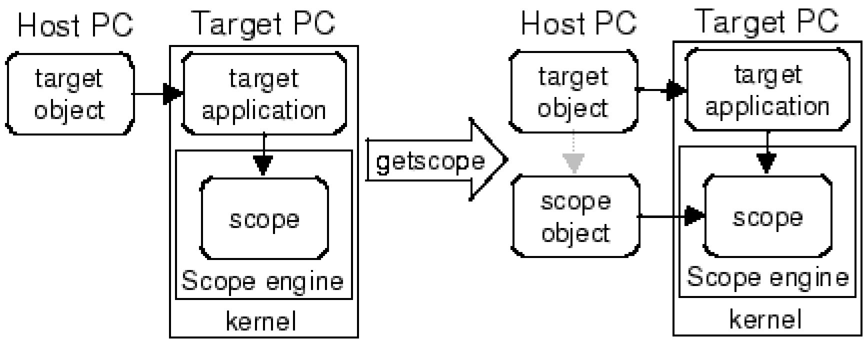
```
scope_object_vector = getscope(target_object, scope_number)
scope_object = target_object.getscope(scope_number)
```

Arguments

`target_object`	Name of a target object.
`scope_number_vector`	Vector of existing scope indices listed in the target object property `Scopes`. The vector can have only one element.
`scope_object`	MATLAB variable for a new scope object vector. The vector can have only one scope object.

Description

`getscope` returns a scope object vector. If you try to get a nonexistent scope, the result is an error. You can retrieve the list of existing scopes using the method `get(target_object, 'scopes')` or `target_object.scopes`.



Examples

If your Simulink model has an Simulink Real-Time scope block, a target scope is created at the time the target application is downloaded to the target computer. To change the number of samples, you must create a scope object and then change the scope object property `NumSamples`.

```
sc1 = getscope(tg,1) or sc1 = tg.getscope(1)
sc1.NumSample = 500
```

The following example gets the properties of all scopes on the target computer and creates a vector of scope objects on the host computer. If the target object has more than one scope, it create a vector of scope objects.

```
scvector = getscope(tg)
```

See Also

`SimulinkRealTime.getTargetSettings` | `SimulinkRealTime.target.remscope`

How To

- “Application and Driver Scripts”

Purpose

Value of target object signal index

Syntax

```
getsignal(target_object, signal index)
```

Arguments

`target_object`	Name of a target object. The default name is `tg`.
`signal_index`	Index number of the signal.

Description

`getsignal` returns the value of the signal associated with `signal_index`.

Examples

Get the value of signal index `2`.

```
getsignal(tg, 2)
ans =  -3.3869e+006
```

SimulinkRealTime.target.getsignalid

Purpose

Signal index or signal property from signal list

Syntax

```
getsignalid(target_object, 'signal_name')
tg.getsignalid('signal_name')
```

Arguments

`target_object`	Name of an existing target object.
`signal_name`	Enter the name of a signal from your Simulink model. For blocks with a single signal, the `signal_name` is equal to the `block_name`. For blocks with multiple signals, the Simulink Real-Time software appends S1, S2 . . . to the `block_name`.

Description

`getsignalid` returns the index or name of a signal from the signal list, based on the path to the signal name. The block names must be entered in full and are case sensitive. Note, enter for `block_name` the mangled name that Simulink Coder uses for code generation.

Examples

Get the signal index for the single signal from the Simulink block `Gain1`.

```
tg = slrt;
getsignalid(tg, 'Gain1') or tg.getsignalid('Gain1')
ans = 6
```

See Also

`SimulinkRealTime.target.getparamid`

How To

- “Application and Driver Scripts”
- “Why Does the getparamid Function Return Nothing?”

SimulinkRealTime.target.getsignalidsfromlabel

Purpose

Return vector of signal indices

Syntax

```
getsignalidsfromlabel(target_object, signal_label)
target_object.getsignalidsfromlabel(signal_label)
```

Arguments

`target_object`	Name of a target object. The default name is `tg`.
`signal_label`	Signal label (from Simulink model).

Description

`getsignalidsfromlabel` returns a vector of one or more signal indices that are associated with the labeled signal, `signal_label`. This function assumes that you have labeled the signal for which you request the index (see the **Signal name** parameter of the “Signal Properties Controls”). Note that the 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;
tg.getsignalidsfromlabel('xpcoscGain')
ans =
0
```

See Also

`SimulinkRealTime.target.getsignallabel`

SimulinkRealTime.target.getsignallabel

Purpose Return signal label

Syntax

```
getsignallabel(target_object, signal_index)
target_object.getsignallabel(signal_index)
```

Arguments

`target_object`	Name of a target object. The default name is `tg`.
`signal_index`	Index number of the signal.

Description `getsignallabel` returns the signal label for the specified signal index, `signal_index`. `signal_label`. This function assumes that you have labeled the signal for which you request the label (see the **Signal name** parameter of the “Signal Properties Controls”). Note that the Simulink Real-Time software refers to Simulink signal names as signal labels.

Examples

```
tg = slrt;
getsignallabel(tg, 0)
ans =
xpcoscGain
```

See Also `SimulinkRealTime.target.getsignalidsfromlabel`

Purpose

Signal name from index list

Syntax

```
getsignalname(target_object, signal_index)
target_object.getsignalname(signal_index)
```

Arguments

`target_object`	Name of a target object. The default name is `tg`.
`signal_index`	Index number of the signal.

Description

`getsignalname` returns one argument string, signal name, from the index list for the specified signal index.

Examples

Get the signal name of signal ID 2.

```
[sigName]=getsignalname(tg,2)
sigName =
Gain2
```

SimulinkRealTime.target.load

Purpose

Download target application to target computer

Syntax

```
target_object = target_object.load(target_application)
target_object = load(target_object,target_application)
```

Description

`target_object = target_object.load(target_application)` loads the application `target_application` onto the target computer represented by `target_object`.

The call returns `target_object`, updated with the new state of the target.

`target_object = load(target_object,target_application)` is an alternative syntax.

Input Arguments

target_object

Object of type `SimulinkRealTime.target` that represents the target computer. Before calling this function, make sure that you have started the target computer with the Simulink Real-Time kernel and have applied the required host-target communication settings.

Data Types
`struct`

target_application

Name of the target application, without file extension. `target_application` can also contain the absolute path to the target application, without file extension.

You must build the application in the current working folder on the host computer. By default, the Simulink Real-Time software calls SimulinkRealTime.target.load automatically after the Simulink Coder build process completes. If a target application was previously loaded, before downloading the new target application, SimulinkRealTime.target.load unloads the old target application.

If you are running the target application in Standalone mode, a call to SimulinkRealTime.target.load has no effect. To load a new application, you must 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`

Examples

Load `xpcosc`

Load the target 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 target application.

```
tg.load('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.

```
tg.start;
```

See Also

`SimulinkRealTime.target.unload`

Related Examples

- "Application and Driver Scripts"

Purpose

Restore parameter values saved in specified file

Syntax

```
loadparamset(target_object,'filename')
target_object.loadparamset('filename')
```

Arguments

`target_object`	Name of an existing target object.
`filename`	Enter the name of the file that contains the saved parameters.

Description

`loadparamset` restores the target application parameter values saved in the file `filename`. This file must be located on a local drive of the target computer. This method assumes that you have a parameter file from a previous run of the `SimulinkRealTime.target.saveparamset` method.

See Also

`SimulinkRealTime.target.saveparamset`

SimulinkRealTime.target.reboot

Purpose

Reboot target computer

Syntax

MATLAB command line

```
reboot(target_object)
```

Target computer command line

```
reboot
```

Arguments

`target_object`	Name of an existing target object.

Description

`reboot` restarts the target computer, and if a target boot disk is still present, the Simulink Real-Time kernel is reloaded.

On the target computer command line, you can use the corresponding command `reboot`.

You can also use this method to restart the target computer back to Windows after removing the target boot disk.

Note This method might not work on some target hardware.

See Also

`SimulinkRealTime.target.load` | `SimulinkRealTime.target.unload`

Purpose

Remove scope from target computer

Syntax

MATLAB command line

```
remscope(target_object, scope_number_vector)
target_object.remscope(scope_number_vector)
remscope(target_object)
target_object.remscope
```

Target computer command line

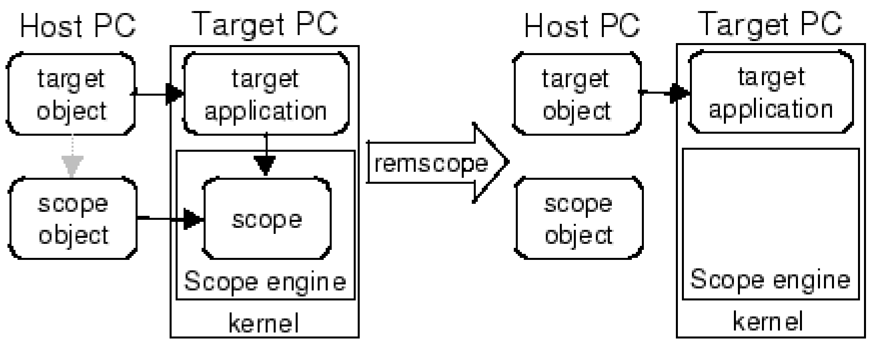
```
remscope scope_number
remscope 'all'
```

Arguments

`target_object`	Name of a target object. The default name is `tg`.
`scope_number_vector`	Vector of existing scope indices listed in the target object property `Scopes`.
`scope_number`	Single scope index.

Description

If a scope index is not given, the method `remscope` deletes all scopes on the target computer. The method `remscope` has no return value. The scope object representing the scope on the host computer is not deleted.



Note that you can only permanently remove scopes that are added with the method `addscope`. This is a scope that is outside a model. If you remove a scope that has been added through a scope block (the scope block is inside the model), a subsequent run of that model creates the scope again.

Examples

Remove a single scope.

```
remscope(tg,1)
```

or

```
tg.remscope(1)
```

Remove two scopes.

```
remscope(tg,[1 2])
```

or

```
tg.remscope([1,2])
```

Remove all scopes.

```
remscope(tg)
```

or

```
tg.remscope
```

See Also

`SimulinkRealTime.target.addscope` | `SimulinkRealTime.target.getscope`

How To

- “Application and Driver Scripts”

SimulinkRealTime.target.saveparamset

Purpose

Save current target application parameter values

Syntax

```
saveparamset(target_object,'filename')
target_object.saveparamset('filename')
```

Arguments

`target_object`	Name of an existing target object.
`filename`	Enter the name of the file to contain the saved parameters.

Description

`saveparamset` saves the target application parameter values in the file `filename`. This method saves the file on a local drive of the target computer (C:\ by default). You can later reload these parameters with the `loadparamset` function.

You might want to save target application parameter values if you change these parameter values while the application is running in real time. Saving these values enables you to easily recreate target application parameter values from a number of application runs.

See Also

`SimulinkRealTime.target.loadparamset`

SimulinkRealTime.target.set

Purpose

Change target application object property values

Syntax

MATLAB command line

```
set(target_object)
set(target_object, 'property_name1', 'property_value1',
'property_name2', 'property_value2', . . .)
target_object.set('property_name1', 'property_value1')
set(target_object, property_name_vector,
property_value_vector)
target_object.property_name = property_value
```

Target computer command line - Commands are limited to the target object properties `stoptime`, `sampletime`, and parameters.

```
parameter_name = parameter_value
stoptime = floating_point_number
sampletime = floating_point_number
```

Arguments

`target_object`	Name of a target object.
`'property_name'`	Name of a target object property. Always use quotation marks.
`property_value`	Value for a target object property. Always use quotation marks for character strings; quotation marks are optional for numbers.

Description

`set` sets the properties of the target object. Not all properties are user writable.

Properties must be entered in pairs or, using the alternate syntax, as one-dimensional cell arrays of the same size. This means they must both be row vectors or both column vectors, and the corresponding values for properties in `property_name_vector` are stored in `property_value_vector`. The writable properties for a target object are listed in the following table. This table includes a description of the properties:

Property	Description	Writable
`Application`	Name of the Simulink model and target application built from that model.	No
`AvgTET`	Average task execution time. This value is an average of the measured CPU times, in seconds, to run the model equations and post outputs during each sample interval. Task execution time is nearly constant, with minor deviations due to cache, memory access, interrupt latency, and multirate model execution. The TET includes: • Complete I/O latency. • Data logging (the parts that happen in a real-time task). This includes data captured in scopes. • Asynchronous interruptions. • Parameter updating latency (if the **Double buffer parameter changes** parameter is set in the **Simulink Real-Time Options** node of the model Configuration Parameters dialog box). Note that the TET is not the only consideration in determining the minimum achievable sample time. Other considerations, not included in the TET, are: • Time required to measure TET	No

Property	Description	Writable
	• Interrupt latency required to schedule and run one step of the model	
`CommunicationTimeOut`	Communication timeout between host and target computer, in seconds.	Yes
`Connected`	Communication status between the host computer and the target computer. Values are `'Yes'` and `'No'`.	No
`CPUoverload`	CPU status for overload. If the target application requires more CPU time than the sample time of the model, this value is set from `'none'` to `'detected'` and the current run is stopped. Returning this status to `'none'` requires either a faster processor or a larger sample time.	No
`ExecTime`	Execution time. Time, in seconds, since your target application started running. When the target application stops, the total execution time is displayed.	No
`LogMode`	Controls which data points are logged: • Time-equidistant logging. Logs a data point at every time interval. Set value to `'Normal'`. • Value-equidistant logging. Logs a data point only when an output signal from the `OutputLog` changes by a specified value (increment). Set the value to the difference in signal values.	Yes

Property	Description	Writable
`MaxLogSamples`	Maximum number of samples for each logged signal within the circular buffers for `TimeLog`, `StateLog`, `OutputLog`, and `TETLog`. `StateLog` and `OutputLog` can have one or more signals. This value is calculated by dividing the **Signal Logging Buffer Size** by the number of logged signals. The **Signal Logging Buffer Size** box is in the **Simulink Real-Time Options** pane of the Configuration Parameters dialog box.	No
`MaxTET`	Maximum task execution time. Corresponds to the slowest time (longest time measured), in seconds, to update model equations and post outputs.	No
`MinTET`	Minimum task execution time. Corresponds to the fastest time (smallest time measured), in seconds, to update model equations and post outputs.	No
`Mode`	Type of Simulink Coder code generation. Values are `'Real-Time Singletasking'`, `'Real-Time Multitasking'`, and `'Accelerate'`. The default value is `'Real-Time Singletasking'`. Even if you select `'Real-Time Multitasking'`, the actual mode can be `'Real-Time Singletasking'`. This happens if your model contains only one or two tasks and the sample rates are equal.	No

Property	Description	Writable
`NumLogWraps`	The number of times the circular buffer wrapped. The buffer wraps each time the number of samples exceeds `MaxLogSamples`.	No
`NumParameters`	The number of parameters from your Simulink model that you can tune or change.	No
`NumSignals`	The number of signals from your Simulink model that are available to be viewed with a scope.	No
`OutputLog`	Storage in the MATLAB workspace for the output or Y-vector logged during execution of the target application.	No
`Parameters`	List of tunable parameters. This list is visible only when `ShowParameters` is set to `'on'`: • Property value. Value of the parameter in a Simulink block. • Type. Data type of the parameter. Always `double`. • Size. Size of the parameter. For example, scalar, 1-by-2 vector, or 2-by-3 matrix. • Parameter name. Name of a parameter in a Simulink block. • Block name. Name of a Simulink block.	No

Property	Description	Writable
`SampleTime`	Time between samples. This value equals the step size, in seconds, for updating the model equations and posting the outputs. (See “Alternative Configuration and Control Methods” for limitations on target property changes to sample times.)	Yes
`Scopes`	List of index numbers, with one index for each scope.	No
`SessionTime`	Time since the kernel started running on your target computer. This is also the elapsed time since you started the target computer. Values are in seconds.	No
`ShowParameters`	Flag set to view or hide the list of parameters from your Simulink blocks. This list is shown when you display the properties for a target object. Values are `'on'` and `'off'`.	Yes
`ShowSignals`	Flag set to view or hide the list of signals from your Simulink blocks. This list is shown when you display the properties for a target object. Values are `'on'` and `'off'`.	Yes
`Signals`	List of viewable signals. This list is visible only when `ShowSignals` is set to `'on'`. • Property name. `S0`, `S1`. . . • Property value. Value of the signal. • Block name. Name of the Simulink block the signal is from.	No

Property	Description	Writable
`StateLog`	Storage in the MATLAB workspace for the state or x-vector logged during execution of the target application.	No
`Status`	Execution status of your target application. Values are `'stopped'` and `'running'`.	No
`StopTime`	Time when the target application stops running. Values are in seconds. The original value is set in the **Solver** pane of the Configuration Parameters dialog box. When the `ExecTime` reaches `StopTime`, the application stops running.	Yes
`TETLog`	Storage in the MATLAB workspace for a vector containing task execution times during execution of the target application. To enable logging of the TET, you must select the **Log Task Execution Time** check box in the **Simulink Real-Time Options** pane of the Configuration Parameters dialog box.	No
`TimeLog`	Storage in the MATLAB workspace for the time or T-vector logged during execution of the target application.	No
`ViewMode`	Display either all scopes or a single scope on the target computer. Value is `'all'` or a single scope index. This property is active only if the environment property `TargetScope` is set to `enabled`.	Yes

The function `set` typically does not return a value. However, if called with an explicit return argument, for example, `a = set(target_object, property_name, property_value)`, it returns the value of the properties after the indicated settings have been made.

Examples

Get a list of writable properties for a scope object.

```
set(tg)
ans =
          StopTime: {}
        SampleTime: {}
          ViewMode: {}
           LogMode: {}
    ShowParameters: {}
       ShowSignals: {}
```

Change the property `ShowSignals` to on.

```
tg.set('showsignals', 'on') or set(tg, 'showsignals', 'on')
```

As an alternative to the method `set`, use the target object property `ShowSignals`. In the MATLAB window, type

```
tg.showsignals ='on'
```

See Also

`SimulinkRealTime.target.get` | `set` | `SimulinkRealTime.fileScope.set` | `SimulinkRealTime.hostScope.set` | `SimulinkRealTime.targetScope.set`

How To

- “Application and Driver Scripts”

SimulinkRealTime.target.setparam

Purpose

Change writable target object parameters

Syntax

```
setparam(target_object, parameter_index, parameter_value)
```

Arguments

`target_object`	Name of an existing target object. The default name is `tg`.
`parameter_index`	Index number of the parameter.
`parameter_value`	Value for a target object parameter.

Description

Method of a target object. Set the value of the target parameter. This method returns a structure that stores the parameter index, previous parameter values, and new parameter values in the following fields:

- `parIndexVec`
- `OldValues`
- `NewValues`

Examples

Set the value of parameter index `5` to `100`.

```
setparam(tg, 5, 100)
ans =
parIndexVec: 5
OldValues: 400
NewValues: 100
```

Simultaneously set values for multiple parameters. Use the cell array format to specify new parameter values.

```
setparam(tg, [1 5],{10,100})
ans =
parIndexVec: [1 5]
OldValues: {[2]  [4]}
NewValues: {[10]  [100]}
```

SimulinkRealTime.target.start

Purpose

Start execution of target application on target computer

Syntax

MATLAB command line

```
start(target_object)
target_object.start
+target_object
```

Target computer command line

```
start
```

Arguments

`target_object`	Name of a target object. The default name is `tg`.

Description

Method of both target objects. Starts execution of the target application represented by the target object. Before using this method, the target application must be created and loaded on the target computer. If a target application is running, this command has no effect.

Examples

Start the target application represented by the target object `tg`.

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

See Also

`SimulinkRealTime.target.stop` | `SimulinkRealTime.target.load` | `SimulinkRealTime.fileScope.stop` | `SimulinkRealTime.hostScope.stop` | `SimulinkRealTime.targetScope.stop`

SimulinkRealTime.target.stop

Purpose

Stop execution of target application on target computer

Syntax

MATLAB command line

```
stop(target_object)
target_object.stop
-target_object
```

Target computer command line

```
stop
```

Arguments

`target_object`	Name of a target object.

Description

Stops execution of the target application represented by the target object. If the target application is stopped, this command has no effect.

Examples

Stop the target application represented by the target object `tg`.

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

See Also

`SimulinkRealTime.target.start | SimulinkRealTime.fileScope.stop | SimulinkRealTime.hostScope.stop | SimulinkRealTime.targetScope.stop`

Purpose

Tests communication between host and target computers

Syntax

```
SimulinkRealTime.target.ping
```

Description

Returns `success` if the Simulink Real-Time kernel is loaded and running, and communication is working between the host and target computers. Otherwise, returns `failed`.

`SimulinkRealTime.target.ping` without an argument returns `success` if the host computer and the target computer can communicate using the settings for that computer. Otherwise, returns `failed`.

Examples

Check communication with default target computer

```
tg = slrt;
tg.ping
```

Check communication with specified target computer

```
tg = slrt('TargetPC1');
tg.ping
```

SimulinkRealTime.target.unload

Purpose

Remove current target application from target computer

Syntax

```
unload(target_object)
target_object.unload
```

Arguments

`target_object`	Name of a target object that represents a target application.

Description

Method of a target object. The kernel goes into loader mode and is ready to download new target application from the host computer.

If you are running in StandAlone mode, this command has no effect. To unload and reload a new application, you must rebuild the standalone application with the new application, then restart the target computer with the updated standalone application.

Examples

Unload the target application represented by the target object `tg`.

```
unload(tg) or tg.unload
```

See Also

`SimulinkRealTime.target.load` | `SimulinkRealTime.target.reboot`

SimulinkRealTime.target.viewTargetScreen

Purpose

Open Real-Time Simulink Real-Time window on host computer

Syntax

`SimulinkRealTime.target.viewTargetScreen`

Description

`SimulinkRealTime.target.viewTargetScreen` opens a Simulink Real-Time display window for `target_object`.

If you have one target computer, or if you designate a target computer as the default one in your system, use the following syntax after you build and download the target application:

```
tg = slrt;
tg.viewTargetScreen
```

If you have multiple target computers in your system, create the target object first:

```
tg = SimulinkRealTime.target('target_name')
tg.viewTargetScreen
```

The behavior of this function depends on the value for the environment property `TargetScope`:

- If `TargetScope` is enabled, a single graphics screen is uploaded.

  The screen is not continually updated because of a higher data volume when a target graphics card is in VGA mode. You must explicitly request an update. To manually update the host screen with another target screen, move the pointer into the 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 host and displayed in the window.

Examples

To open the Simulink Real-Time display window for the default target computer in the Command Window, type:

```
tg = slrt;
tg.viewTargetScreen
```

To open the display window for target computer `TargetPC1` in the Command Window, type:

```
tg1 = slrt('TargetPC1');
tg1.viewTargetScreen
```

Purpose

Control and access properties of file scopes

Description

The scope gets a data package from the kernel and stores the data in a file in the target computer file system. Depending on the setting of `WriteMode`, the file size is or is not continuously updated. You can then transfer the data to another computer for examination or plotting.

Methods

These methods are held in common by file, host, and target scopes.

Method	Description
`SimulinkRealTime.fileScope.addsignal`	Add signals to scope represented by scope object
`SimulinkRealTime.fileScope.get`	Return property values for scope object
`SimulinkRealTime.fileScope.remsignal`	Remove signals from scope represented by scope object
`SimulinkRealTime.fileScope.set`	Change property values for scope object
`SimulinkRealTime.fileScope.start`	Start execution of scope on target computer
`SimulinkRealTime.fileScope.stop`	Stop execution of scope on target computer
`SimulinkRealTime.fileScope.trigger`	Software-trigger start of data acquisition for scope or scopes

Properties

These properties are held in common by file, host, and target scopes.

Property	Description	Writable
`Application`	Name of the Simulink model associated with this scope object.	No
`Decimation`	A number `n`, where every `n`th sample is acquired in a scope window.	Yes

Property	Description	Writable
`NumPrePostSamples`	Number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set `TriggerMode` to `'FreeRun'`, this property has no effect on data acquisition.	Yes
`NumSamples`	Number of contiguous samples captured during the acquisition of a data package. If the scope stops before capturing this number of samples, the scope has the collected data up to the end of data collection, then has zeroes for the remaining uncollected data. Note that you should know what type of data you are collecting, it is possible that your data contains zeroes. For file scopes, this parameter works in conjunction with the **AutoRestart** check box. If the **AutoRestart** box is selected, the file scope collects data up to **Number of Samples**, then starts over again, overwriting the buffer. If the **AutoRestart** box is not selected, the file scope collects data only up to **Number of Samples**, then stops.	Yes
`ScopeId`	A numeric index, unique for each scope.	No
`Signals`	List of signal indices from the target object to display on the scope.	Yes
`Status`	Indicate whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are `'Acquiring'`, `'Ready for being Triggered'`, `'Interrupted'`, and `'Finished'`.	No

Property	Description	Writable
TriggerLevel	If TriggerMode is 'Signal', indicates the value the signal has to cross to trigger the scope and start acquiring data. The trigger level can be crossed with either a rising or falling signal.	Yes
TriggerMode	Trigger mode for a scope. Valid values are 'FreeRun' (default), 'Software', 'Signal', and 'Scope'.	Yes
TriggerSample	If TriggerMode is 'Scope', then TriggerSample specifies which sample of the triggering scope the current scope should trigger on. For example, if TriggerSample is 0 (default), the current scope triggers on sample 0 (first sample acquired) of the triggering scope. This means that the two scopes will be perfectly synchronized. If TriggerSample is 1, the first sample (sample 0) of the current scope will be at the same instant as sample number 1 (second sample in the acquisition cycle) of the triggering scope. As a special case, setting TriggerSample to -1 means that the current scope is triggered at the end of the acquisition cycle of the triggering scope. Thus, the first sample of the triggering scope is acquired one sample after the last sample of the triggering scope.	Yes
TriggerScope	If TriggerMode is 'Scope', identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered. You do this by setting the slave scope property TriggerScope to the scope index of the master scope.	Yes

Property	Description	Writable
TriggerSignal	If TriggerMode is 'Signal', identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property Signal.	Yes
TriggerSlope	If TriggerMode is 'Signal', indicates whether the trigger is on a rising or falling signal. Values are 'Either' (default), 'Rising', and 'Falling'.	Yes
Type	Determines whether the scope is displayed on the host computer or on the target computer. Values are 'Host', 'Target', and 'File'. Property Type is set only once, when the scope is created on the target computer.	No

These properties are specific to file scopes.

Property	Description	Writeable
AutoRestart	Values are 'on' and 'off'. For file scopes, enable the file scope to collect data up to the number of samples (NumSamples), then start over again, appending the new data to the end of the signal data file. Clear the **AutoRestart** check box to have the file scope collect data up to **Number of samples**, then stop. If the named signal data file already exists when you start the target application, the software overwrites the old data with the new signal data.	No

Property	Description	Writeable
	To use the DynamicFileName property, set AutoRestart to 'on' first. For host or target scopes, this parameter has no effect.	
DynamicFileName	Values are 'on' and 'off'. By default, the value is 'off'. Enable the ability to dynamically create multiple log files for file scopes. To use DynamicFileName, set AutoRestart to 'on' first. When you enable DynamicFileName, configure Filename to create incrementally numbered file names for the multiple log files. Failure to do so causes an error when you try to start the scope. You can enable the creation of up to 99999999 files (<%%%%%%%%>.dat). The length of a file name, including the specifier, cannot exceed eight characters. For host or 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. Note that for file scopes created through the MATLAB interface, no name is initially assigned to `FileName`. After you start the scope, the software assigns a name for the file to acquire the signal data. This name typically consists of the scope object name, `ScopeId`, and the beginning letters of the first signal added to the scope.
	
	If you set `DynamicFileName` and `AutoRestart` to `'on'`, configure `Filename` to dynamically increment. Use a base file name, an underscore (_), and a `< >` specifier. Within the specifier, enter one to eight `%` symbols. Each symbol `%` represents a decimal location in the file name. The specifier can appear anywhere in the file name. For example, the following value for `Filename`, `C:\work\file_<%%%>.dat` creates file names with the following pattern:
	
	`file_001.dat`
	`file_002.dat`
	`file_003.dat`
		No

Property	Description	Writeable
	The last file name of this series will be `file_999.dat`. If the function is still logging data when the last file name reaches its maximum size, the function starts from the beginning and overwrites the first file name in the series. If you do not retrieve the data from existing files before they are overwritten, the data is lost. For host or target scopes, this parameter has no effect.	
`MaxWriteFileSize`	Provide the maximum size of `Filename`, in bytes. This value must be a multiple of `WriteSize`. Default is `536870912`. When the size of a log file reaches `MaxWriteFileSize`, the software creates a subsequently numbered file name, and continues logging data to that file, up until the highest log file number you have specified. If the software cannot create additional log files, it overwrites the first log file. For host or target scopes, this parameter has no effect.	Yes

Property	Description	Writeable
`Mode`	**Note** The `Mode` property will be removed in a future release. • For target scopes, use `DisplayMode`. • For file scopes, use `WriteMode`. • For host scopes, this parameter has no effect.	Yes
`WriteMode`	For file scopes, specify when a file allocation table (FAT) entry is updated. Values are `'Lazy'` or `'Commit'`. Both modes write the signal data to the file. With `'Commit'` mode, each file write operation simultaneously updates the FAT entry for the file. This mode is slower, but the file system maintains the actual file size. With `'Lazy'` mode, the FAT entry is updated only when the file is closed and not during each file write operation. This mode is faster, but if the system crashes before the file is closed, the file system might not know the actual file size (the file contents, however, will be intact). For host or target scopes, this parameter has no effect.	Yes
`WriteSize`	Enter the block size, in bytes, of the data chunks. This parameter	Yes

Property	Description	Writeable
	specifies that a memory buffer, of length number of samples (`NumSamples`), collect data in multiples of `WriteSize`. By default, this parameter is 512 bytes, which is the typical disk sector size. Using a block size that is the same as the disk sector size provides better performance. If you experience a system crash, you can expect to lose an amount of data the size of `WriteSize`. For host or target scopes, this parameter has no effect.	

SimulinkRealTime.fileScope.addsignal

Purpose

Add signals to scope represented by scope object

Syntax

MATLAB command line

```
addsignal(scope_object_vector, signal_index_vector)
scope_object_vector.addsignal(signal_index_vector)
```

Target command line

```
addsignal scope_index = signal_index, signal_index, . . .
```

Arguments

`scope_object_vector`	Name of a single scope object or the name of a vector of scope objects.
`signal_index_vector`	For one signal, use a single number. For two or more signals, enclose numbers in brackets and separate with commas.
`scope_index`	Single scope index.

Description

`addsignal` adds signals to a scope object. The signals must be specified by their indices, which you can retrieve using the target object method `getsignalid`. If the `scope_object_vector` has two or more scope objects, the same signals are assigned to each scope.

Note You must stop the scope before you can add a signal to it.

Examples

Add signals `0` and `1` from the target object `tg` to the scope object `sc1`. The signals are added to the scope, and the scope object property `Signals` is updated to include the added signals.

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

Display a list of properties and values for the scope object `sc1` with the property `Signals`, as shown below.

```
sc1.Signals
Signals              = 1  : Signal Generator
                       0  : Integrator1
```

Another way to add signals without using the method `addsignal` is to use the scope object method `set`.

```
set(sc1,'Signals', [0,1]) or sc1.set('signals',[0,1]
```

Or, to directly assign signal values to the scope object property `Signals`,

```
sc1.signals = [0,1]
```

See Also

`SimulinkRealTime.fileScope.remsignal | SimulinkRealTime.fileScope.set | SimulinkRealTime.target.addscope | SimulinkRealTime.target.getsignalid`

How To

- "Target Scope Usage"
- "Host Scope Usage"
- "File Scope Usage"
- "Application and Driver Scripts"

SimulinkRealTime.fileScope.get

Purpose

Return property values for scope objects

Syntax

```
get(scope_object_vector)
get(scope_object_vector, 'scope_object_property')
get(scope_object_vector, scope_object_property_vector)
```

Arguments

`target_object`	Name of a target object.
`scope_object_vector`	Name of a single scope or name of a vector of scope objects.
`scope_object_property`	Name of a scope object property.

Description

`get` gets the value of readable scope object properties from a scope object or the same property from each scope object in a vector of scope objects. Scope object properties let you select signals to acquire, set triggering modes, and access signal information from the target application. You can view and change these properties using scope object methods.

The properties for a scope object are listed in the following table. This table includes descriptions of the properties and the properties you can change directly by assigning a value.

Property	Description	Writable
`Application`	Name of the Simulink model associated with this scope object.	No
`Decimation`	A number `n`, where every `n`th sample is acquired in a scope window.	Yes
`NumPrePostSamples`	Number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set `TriggerMode` to `'FreeRun'`, this property has no effect on data acquisition.	Yes

Property	Description	Writable
`NumSamples`	Number of contiguous samples captured during the acquisition of a data package. If the scope stops before capturing this number of samples, the scope has the collected data up to the end of data collection, then has zeroes for the remaining uncollected data. Note that you should know what type of data you are collecting, it is possible that your data contains zeroes. For file scopes, this parameter works in conjunction with the **AutoRestart** check box. If the **AutoRestart** box is selected, the file scope collects data up to **Number of Samples**, then starts over again, overwriting the buffer. If the **AutoRestart** box is not selected, the file scope collects data only up to **Number of Samples**, then stops.	Yes
`ScopeId`	A numeric index, unique for each scope.	No
`Signals`	List of signal indices from the target object to display on the scope.	Yes
`Status`	Indicate whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are `'Acquiring'`, `'Ready for being Triggered'`, `'Interrupted'`, and `'Finished'`.	No
`TriggerLevel`	If `TriggerMode` is `'Signal'`, indicates the value the signal has to cross to trigger the scope and start acquiring data. The trigger level can be crossed with either a rising or falling signal.	Yes
`TriggerMode`	Trigger mode for a scope. Valid values are `'FreeRun'` (default), `'Software'`, `'Signal'`, and `'Scope'`.	Yes

Property	Description	Writable
`TriggerSample`	If `TriggerMode` is `'Scope'`, then `TriggerSample` specifies which sample of the triggering scope the current scope should trigger on. For example, if `TriggerSample` is `0` (default), the current scope triggers on sample `0` (first sample acquired) of the triggering scope. This means that the two scopes will be perfectly synchronized. If `TriggerSample` is `1`, the first sample (sample `0`) of the current scope will be at the same instant as sample number 1 (second sample in the acquisition cycle) of the triggering scope. As a special case, setting `TriggerSample` to `-1` means that the current scope is triggered at the end of the acquisition cycle of the triggering scope. Thus, the first sample of the triggering scope is acquired one sample after the last sample of the triggering scope.	Yes
`TriggerScope`	If `TriggerMode` is `'Scope'`, identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered. You do this by setting the slave scope property `TriggerScope` to the scope index of the master scope.	Yes
`TriggerSignal`	If `TriggerMode` is `'Signal'`, identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property `Signal`.	Yes

Property	Description	Writable
`TriggerSlope`	If `TriggerMode` is `'Signal'`, indicates whether the trigger is on a rising or falling signal. Values are `'Either'` (default), `'Rising'`, and `'Falling'`.	Yes
`Type`	Determines whether the scope is displayed on the host computer or on the target computer. Values are `'Host'`, `'Target'`, and `'File'`. Property `Type` is set only once, when the scope is created on the target computer.	No

Examples

List the readable properties, along with their current values. This is given in the form of a structure whose field names are the property names and whose field values are property values.

```
get(sc)
```

List the value for the scope object property `Type`. Notice that the property name is a string, in quotation marks, and is not case sensitive.

```
get(sc,'type')
ans = Target
```

See Also

`SimulinkRealTime.fileScope.set` | `SimulinkRealTime.hostScope.set` | `SimulinkRealTime.targetScope.set` | `get` | `SimulinkRealTime.target.get`

SimulinkRealTime.fileScope.remsignal

Purpose

Remove signals from scope represented by scope object

Syntax

MATLAB command line

```
remsignal(scope_object)
remsignal(scope_object, signal_index_vector)
scope_object.remsignal(signal_index_vector)
```

Target command line

```
remsignal scope_index = signal_index, signal_index, . . .
```

Arguments

`scope_object`	MATLAB object created with the target object method `addscope` or `getscope`.
`signal_index_vector`	Index numbers from the scope object property `Signals`. This argument is optional, and if it is left out all signals are removed.
`signal_index`	Single signal index.

Description

`remsignal` removes signals from a scope object. The signals must be specified by their indices, which you can retrieve using the target object method `getsignalid`. If the `scope_index_vector` has two or more scope objects, the same signals are removed from each scope. The argument `signal_index` is optional; if it is left out, all signals are removed.

Note You must stop the scope before you can remove a signal from it.

Examples

Remove signals 0 and 1 from the scope represented by the scope object sc1.

```
sc1.get('signals')
ans= 0 1
```

Remove signals from the scope on the target computer with the scope object property `Signals` updated.

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

or

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

See Also

`SimulinkRealTime.fileScope.addsignal` | `SimulinkRealTime.hostScope.addsignal` | `SimulinkRealTime.targetScope.addsignal` | `SimulinkRealTime.target.getsignalid`

SimulinkRealTime.fileScope.set

Purpose

Change property values for scope objects

Syntax

```
set(scope_object_vector)
set(scope_object_vector, property_name1, property_value1,
property_name2, property_value2, . . .)
scope_object_vector.set('property_name1', property_value1,
. . .)
set(scope_object, 'property_name', property_value, . . .)
```

Arguments

`scope_object`	Name of a scope object or a vector of scope objects.
`'property_name'`	Name of a scope object property. Always use quotation marks.
`property_value`	Value for a scope object property. Always use quotation marks for character strings; quotation marks are optional for numbers.

Description

Method for scope objects. Sets the properties of the scope object. Not all properties are user writable. Scope object properties let you select signals to acquire, set triggering modes, and access signal information from the target application. You can view and change these properties using scope object methods.

Properties must be entered in pairs or, using the alternate syntax, as one-dimensional cell arrays of the same size. This means they must both be row vectors or both column vectors, and the corresponding values for properties in `property_name_vector` are stored in `property_value_vector`.

The function `set` typically does not return a value. However, if called with an explicit return argument, for example, `a = set(target_object, property_name, property_value)`, it returns the values of the properties after the indicated settings have been made.

The properties for a scope object are listed in the following table. This table includes descriptions of the properties and the properties you can change directly by assigning a value.

Property	Description	Writable
`Application`	Name of the Simulink model associated with this scope object.	No
`Decimation`	A number `n`, where every `n`th sample is acquired in a scope window.	Yes
`NumPrePostSamples`	Number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set `TriggerMode` to `'FreeRun'`, this property has no effect on data acquisition.	Yes
`NumSamples`	Number of contiguous samples captured during the acquisition of a data package. If the scope stops before capturing this number of samples, the scope has the collected data up to the end of data collection, then has zeroes for the remaining uncollected data. Note that you should know what type of data you are collecting, it is possible that your data contains zeroes. For file scopes, this parameter works in conjunction with the **AutoRestart** check box. If the **AutoRestart** box is selected, the file scope collects data up to **Number of Samples**, then starts over again, overwriting the buffer. If the **AutoRestart** box is not selected, the file scope collects data only up to **Number of Samples**, then stops.	Yes
`ScopeId`	A numeric index, unique for each scope.	No
`Signals`	List of signal indices from the target object to display on the scope.	Yes

Property	Description	Writable
`Status`	Indicate whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are `'Acquiring'`, `'Ready for being Triggered'`, `'Interrupted'`, and `'Finished'`.	No
`TriggerLevel`	If `TriggerMode` is `'Signal'`, indicates the value the signal has to cross to trigger the scope and start acquiring data. The trigger level can be crossed with either a rising or falling signal.	Yes
`TriggerMode`	Trigger mode for a scope. Valid values are `'FreeRun'` (default), `'Software'`, `'Signal'`, and `'Scope'`.	Yes
`TriggerSample`	If `TriggerMode` is `'Scope'`, then `TriggerSample` specifies which sample of the triggering scope the current scope should trigger on. For example, if `TriggerSample` is `0` (default), the current scope triggers on sample `0` (first sample acquired) of the triggering scope. This means that the two scopes will be perfectly synchronized. If `TriggerSample` is `1`, the first sample (sample `0`) of the current scope will be at the same instant as sample number 1 (second sample in the acquisition cycle) of the triggering scope. As a special case, setting `TriggerSample` to `-1` means that the current scope is triggered at the end of the acquisition cycle of the triggering scope. Thus, the first sample of the triggering scope is acquired one sample after the last sample of the triggering scope.	Yes

Property	Description	Writable
`TriggerScope`	If `TriggerMode` is `'Scope'`, identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered. You do this by setting the slave scope property `TriggerScope` to the scope index of the master scope.	Yes
`TriggerSignal`	If `TriggerMode` is `'Signal'`, identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property `Signal`.	Yes
`TriggerSlope`	If `TriggerMode` is `'Signal'`, indicates whether the trigger is on a rising or falling signal. Values are `'Either'` (default), `'Rising'`, and `'Falling'`.	Yes
`Type`	Determines whether the scope is displayed on the host computer or on the target computer. Values are `'Host'`, `'Target'`, and `'File'`. Property `Type` is set only once, when the scope is created on the target computer.	No

Examples

Get a list of writable properties for a scope object.

```
sc1 = getscope(tg,1)
set(sc1)
ans=
           NumSamples: {}
           Decimation: {}
          TriggerMode: {5x1 cell}
        TriggerSignal: {}
         TriggerLevel: {}
         TriggerSlope: {4x1 cell}
         TriggerScope: {}
        TriggerSample: {}
              Signals: {}
    NumPrePostSamples: {}
```

```
          Mode: {5x1 cell}
        YLimit: {}
          Grid: {}
```

The property value for the scope object `sc1` is changed to `on`:

```
sc1.set('grid', 'on') or set(sc1, 'grid', 'on')
```

See Also

`set` | `SimulinkRealTime.fileScope.get` | `SimulinkRealTime.hostScope.get` | `SimulinkRealTime.targetScope.get` | `SimulinkRealTime.target.set`

SimulinkRealTime.fileScope.start

Purpose

Start execution of scope on target computer

Syntax

MATLAB command line

```
start(scope_object_vector)
scope_object_vector.start
+scope_object_vector
start(getscope((target_object, signal_index_vector))
```

Target computer command line

```
startscope scope_index
startscope 'all'
```

Arguments

`target_object`	Name of a target object.
`scope_object_vector`	Name of a single scope object, name of vector of scope objects, list of scope object names in vector form `[scope_object1, scope_object2]`, or the target object method `getscope`, which returns a `scope_object` vector.
`signal_index_vector`	Index for a single scope or list of scope indices in vector form.
`scope_index`	Single scope index.

Description

Method for a scope object. Starts a scope on the target computer represented by a scope object on the host computer. This method might not start data acquisition, which depends on the trigger settings. Before using this method, you must create a scope. To create a scope, use the target object method `addscope` or add Simulink Real-Time scope blocks to your Simulink model.

Examples

Start one scope with the scope object sc1.

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

or type

```
start(getscope(tg,1))
```

Start two scopes.

```
somescopes = getscope(tg,[1,2]) or somescopes=
tg.getscope([1,2])
start(somescopes) or somescopes.start
```

or type

```
sc1 = getscope(tg,1) or sc1 =tg.getscope(1)
sc2 = getscope(tg,2) or sc2 = tg.getscope(2)
start([sc1,sc2])
```

or type

```
start(getscope(tg,[1,2])
```

Start all scopes:

```
allscopes = getscope(tg) or allscopes = tg.getscope
start(allscopes) or allscopes.start or +allscopes
```

or type

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

See Also

`SimulinkRealTime.fileScope.stop` | `SimulinkRealTime.hostScope.stop` | `SimulinkRealTime.targetScope.stop` | `SimulinkRealTime.target.getscope` | `SimulinkRealTime.target.start`

SimulinkRealTime.fileScope.stop

Purpose

Stop execution of scope on target computer

Syntax

MATLAB command line

```
stop(scope_object_vector)
scope_object.stop
-scope_object
stop(getscope(target_object, signal_index_vector))
```

Target computer command line

```
stopscope scope_index
stopscope 'all'
```

Arguments

`target_object`	Name of a target object.
`scope_object_vector`	Name of a single scope object, name of vector of scope objects, list of scope object names in a vector form `[scope_object1, scope_object2]`, or the target object method `getscope`, which returns a `scope_object` vector.
`signal_index_vector`	Index for a single scope or list of scope indices in vector form.
`scope_index`	Single scope index.

Description

Method for scope objects. Stops the scopes represented by the scope objects.

Examples

Stop one scope represented by the scope object `sc1`.

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

Stop all scopes with a scope object vector `allscopes` created with the command

```
allscopes = getscope(tg) or allscopes = tg.getscope.
stop(allscopes) or allscopes.stop or -allscopes
```

or type

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

See Also

SimulinkRealTime.fileScope.start | SimulinkRealTime.hostScope.start | SimulinkRealTime.targetScope.start | SimulinkRealTime.target.getscope | SimulinkRealTime.target.stop

SimulinkRealTime.fileScope.trigger

Purpose

Software-trigger start of data acquisition for scope(s)

Syntax

`trigger(scope_object_vector) or scope_object_vector.trigger`

Arguments

`scope_object_vector`	Name of a single scope object, name of a vector of scope objects, list of scope object names in a vector form `[scope_object1, scope_object2]`, or the target object method `getscope`, which returns a `scope_object` vector.

Description

Method for a scope object. If the scope object property `TriggerMode` has a value of `'software'`, this function triggers the scope represented by the scope object to acquire the number of data points in the scope object property `NumSamples`.

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

Examples

Set a single scope to software trigger, trigger the acquisition of one set of samples, and plot data.

```
sc1 = tg.addscope('host',1) or sc1=addscope(tg,'host',1)
sc1.triggermode = 'software'
tg.start, or start(tg), or +tg
sc1.start  or start(sc1) or +sc1
sc1.trigger or trigger(sc1)
plot(sc1.time, sc1.data)
sc1.stop or stop(sc1) or -sc1
tg.stop or stop(tg) or -tg1
```

Set all scopes to software trigger and trigger to start.

```
allscopes = tg.getscopes
allscopes.triggermode = 'software'
allscopes.start or start(allscopes) or +allscopes
allscopes.trigger or trigger(allscopes)
```

Purpose

Control and access properties of host scopes

Description

The scope gets a data package from the kernel, waits for an upload command from the host computer, and uploads the data to the host. The host computer displays the data using a scope viewer or other MATLAB functions.

Methods

These methods are held in common by file, host, and target scopes.

Method	Description
`SimulinkRealTime.hostScope.addsignal`	Add signals to scope represented by scope object
`SimulinkRealTime.hostScope.get`	Return property values for scope object
`SimulinkRealTime.hostScope.remsignal`	Remove signals from scope represented by scope object
`SimulinkRealTime.hostScope.set`	Change property values for scope object
`SimulinkRealTime.hostScope.start`	Start execution of scope on target computer
`SimulinkRealTime.hostScope.stop`	Stop execution of scope on target computer
`SimulinkRealTime.hostScope.trigger`	Software-trigger start of data acquisition for scope or scopes

Properties

These properties are held in common by file, host, and target scopes.

Property	Description	Writable
`Application`	Name of the Simulink model associated with this scope object.	No
`Decimation`	A number `n`, where every `n`th sample is acquired in a scope window.	Yes

Property	Description	Writable
`NumPrePostSamples`	Number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set `TriggerMode` to `'FreeRun'`, this property has no effect on data acquisition.	Yes
`NumSamples`	Number of contiguous samples captured during the acquisition of a data package. If the scope stops before capturing this number of samples, the scope has the collected data up to the end of data collection, then has zeroes for the remaining uncollected data. Note that you should know what type of data you are collecting, it is possible that your data contains zeroes. For file scopes, this parameter works in conjunction with the **AutoRestart** check box. If the **AutoRestart** box is selected, the file scope collects data up to **Number of Samples**, then starts over again, overwriting the buffer. If the **AutoRestart** box is not selected, the file scope collects data only up to **Number of Samples**, then stops.	Yes
`ScopeId`	A numeric index, unique for each scope.	No
`Signals`	List of signal indices from the target object to display on the scope.	Yes
`Status`	Indicate whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are `'Acquiring'`, `'Ready for being Triggered'`, `'Interrupted'`, and `'Finished'`.	No

Property	Description	Writable
`TriggerLevel`	If `TriggerMode` is `'Signal'`, indicates the value the signal has to cross to trigger the scope and start acquiring data. The trigger level can be crossed with either a rising or falling signal.	Yes
`TriggerMode`	Trigger mode for a scope. Valid values are `'FreeRun'` (default), `'Software'`, `'Signal'`, and `'Scope'`.	Yes
`TriggerSample`	If `TriggerMode` is `'Scope'`, then `TriggerSample` specifies which sample of the triggering scope the current scope should trigger on. For example, if `TriggerSample` is `0` (default), the current scope triggers on sample `0` (first sample acquired) of the triggering scope. This means that the two scopes will be perfectly synchronized. If `TriggerSample` is `1`, the first sample (sample `0`) of the current scope will be at the same instant as sample number 1 (second sample in the acquisition cycle) of the triggering scope. As a special case, setting `TriggerSample` to `-1` means that the current scope is triggered at the end of the acquisition cycle of the triggering scope. Thus, the first sample of the triggering scope is acquired one sample after the last sample of the triggering scope.	Yes
`TriggerScope`	If `TriggerMode` is `'Scope'`, identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered. You do this by setting the slave scope property `TriggerScope` to the scope index of the master scope.	Yes

Property	Description	Writable
TriggerSignal	If TriggerMode is 'Signal', identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property Signal.	Yes
TriggerSlope	If TriggerMode is 'Signal', indicates whether the trigger is on a rising or falling signal. Values are 'Either' (default), 'Rising', and 'Falling'.	Yes
Type	Determines whether the scope is displayed on the host computer or on the target computer. Values are 'Host', 'Target', and 'File'. Property Type is set only once, when the scope is created on the target computer.	No

These properties are specific 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

SimulinkRealTime.hostScope.addsignal

Purpose

Add signals to scope represented by scope object

Syntax

MATLAB command line

```
addsignal(scope_object_vector, signal_index_vector)
scope_object_vector.addsignal(signal_index_vector)
```

Target command line

```
addsignal scope_index = signal_index, signal_index, . . .
```

Arguments

`scope_object_vector`	Name of a single scope object or the name of a vector of scope objects.
`signal_index_vector`	For one signal, use a single number. For two or more signals, enclose numbers in brackets and separate with commas.
`scope_index`	Single scope index.

Description

`addsignal` adds signals to a scope object. The signals must be specified by their indices, which you can retrieve using the target object method `getsignalid`. If the `scope_object_vector` has two or more scope objects, the same signals are assigned to each scope.

Note You must stop the scope before you can add a signal to it.

Examples

Add signals `0` and `1` from the target object `tg` to the scope object `sc1`. The signals are added to the scope, and the scope object property `Signals` is updated to include the added signals.

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

Display a list of properties and values for the scope object `sc1` with the property `Signals`, as shown below.

```
sc1.Signals
Signals              = 1  : Signal Generator
                       0  : Integrator1
```

Another way to add signals without using the method `addsignal` is to use the scope object method `set`.

```
set(sc1,'Signals', [0,1]) or sc1.set('signals',[0,1]
```

Or, to directly assign signal values to the scope object property `Signals`,

```
sc1.signals = [0,1]
```

See Also

`SimulinkRealTime.fileScope.remsignal | SimulinkRealTime.targetScope.set | SimulinkRealTime.target.addscope | SimulinkRealTime.target.getsignalid`

How To

- "Target Scope Usage"
- "Host Scope Usage"
- "File Scope Usage"
- "Application and Driver Scripts"

Purpose

Return property values for scope objects

Syntax

```
get(scope_object_vector)
get(scope_object_vector, 'scope_object_property')
get(scope_object_vector, scope_object_property_vector)
```

Arguments

`target_object`	Name of a target object.
`scope_object_vector`	Name of a single scope or name of a vector of scope objects.
`scope_object_property`	Name of a scope object property.

Description

`get` gets the value of readable scope object properties from a scope object or the same property from each scope object in a vector of scope objects. Scope object properties let you select signals to acquire, set triggering modes, and access signal information from the target application. You can view and change these properties using scope object methods.

The properties for a scope object are listed in the following table. This table includes descriptions of the properties and the properties you can change directly by assigning a value.

Property	Description	Writable
`Application`	Name of the Simulink model associated with this scope object.	No
`Decimation`	A number `n`, where every `n`th sample is acquired in a scope window.	Yes
`NumPrePostSamples`	Number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set `TriggerMode` to `'FreeRun'`, this property has no effect on data acquisition.	Yes

Property	Description	Writable
`NumSamples`	Number of contiguous samples captured during the acquisition of a data package. If the scope stops before capturing this number of samples, the scope has the collected data up to the end of data collection, then has zeroes for the remaining uncollected data. Note that you should know what type of data you are collecting, it is possible that your data contains zeroes. For file scopes, this parameter works in conjunction with the **AutoRestart** check box. If the **AutoRestart** box is selected, the file scope collects data up to **Number of Samples**, then starts over again, overwriting the buffer. If the **AutoRestart** box is not selected, the file scope collects data only up to **Number of Samples**, then stops.	Yes
`ScopeId`	A numeric index, unique for each scope.	No
`Signals`	List of signal indices from the target object to display on the scope.	Yes
`Status`	Indicate whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are `'Acquiring'`, `'Ready for being Triggered'`, `'Interrupted'`, and `'Finished'`.	No
`TriggerLevel`	If `TriggerMode` is `'Signal'`, indicates the value the signal has to cross to trigger the scope and start acquiring data. The trigger level can be crossed with either a rising or falling signal.	Yes
`TriggerMode`	Trigger mode for a scope. Valid values are `'FreeRun'` (default), `'Software'`, `'Signal'`, and `'Scope'`.	Yes

Property	Description	Writable
TriggerSample	If TriggerMode is 'Scope', then TriggerSample specifies which sample of the triggering scope the current scope should trigger on. For example, if TriggerSample is 0 (default), the current scope triggers on sample 0 (first sample acquired) of the triggering scope. This means that the two scopes will be perfectly synchronized. If TriggerSample is 1, the first sample (sample 0) of the current scope will be at the same instant as sample number 1 (second sample in the acquisition cycle) of the triggering scope. As a special case, setting TriggerSample to -1 means that the current scope is triggered at the end of the acquisition cycle of the triggering scope. Thus, the first sample of the triggering scope is acquired one sample after the last sample of the triggering scope.	Yes
TriggerScope	If TriggerMode is 'Scope', identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered. You do this by setting the slave scope property TriggerScope to the scope index of the master scope.	Yes
TriggerSignal	If TriggerMode is 'Signal', identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property Signal.	Yes

Property	Description	Writable
`TriggerSlope`	If `TriggerMode` is `'Signal'`, indicates whether the trigger is on a rising or falling signal. Values are `'Either'` (default), `'Rising'`, and `'Falling'`.	Yes
`Type`	Determines whether the scope is displayed on the host computer or on the target computer. Values are `'Host'`, `'Target'`, and `'File'`. Property `Type` is set only once, when the scope is created on the target computer.	No

Examples

List the readable properties, along with their current values. This is given in the form of a structure whose field names are the property names and whose field values are property values.

```
get(sc)
```

List the value for the scope object property `Type`. Notice that the property name is a string, in quotation marks, and is not case sensitive.

```
get(sc,'type')
ans = Target
```

See Also

`SimulinkRealTime.fileScope.set` |
`SimulinkRealTime.targetScope.set` | `get` |
`SimulinkRealTime.target.get`

SimulinkRealTime.hostScope.remsignal

Purpose

Remove signals from scope represented by scope object

Syntax

MATLAB command line

```
remsignal(scope_object)
remsignal(scope_object, signal_index_vector)
scope_object.remsignal(signal_index_vector)
```

Target command line

```
remsignal scope_index = signal_index, signal_index, . . .
```

Arguments

`scope_object`	MATLAB object created with the target object method `addscope` or `getscope`.
`signal_index_vector`	Index numbers from the scope object property `Signals`. This argument is optional, and if it is left out all signals are removed.
`signal_index`	Single signal index.

Description

`remsignal` removes signals from a scope object. The signals must be specified by their indices, which you can retrieve using the target object method `getsignalid`. If the `scope_index_vector` has two or more scope objects, the same signals are removed from each scope. The argument `signal_index` is optional; if it is left out, all signals are removed.

Note You must stop the scope before you can remove a signal from it.

Examples

Remove signals 0 and 1 from the scope represented by the scope object sc1.

```
sc1.get('signals')
ans= 0 1
```

Remove signals from the scope on the target computer with the scope object property `Signals` updated.

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

or

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

See Also

`SimulinkRealTime.fileScope.addsignal` | `SimulinkRealTime.targetScope.addsignal` | `SimulinkRealTime.target.getsignalid`

Purpose

Change property values for scope objects

Syntax

```
set(scope_object_vector)
set(scope_object_vector, property_name1, property_value1,
property_name2, property_value2, . . .)
scope_object_vector.set('property_name1', property_value1,
. . .)
set(scope_object, 'property_name', property_value, . . .)
```

Arguments

`scope_object`	Name of a scope object or a vector of scope objects.
`'property_name'`	Name of a scope object property. Always use quotation marks.
`property_value`	Value for a scope object property. Always use quotation marks for character strings; quotation marks are optional for numbers.

Description

Method for scope objects. Sets the properties of the scope object. Not all properties are user writable. Scope object properties let you select signals to acquire, set triggering modes, and access signal information from the target application. You can view and change these properties using scope object methods.

Properties must be entered in pairs or, using the alternate syntax, as one-dimensional cell arrays of the same size. This means they must both be row vectors or both column vectors, and the corresponding values for properties in `property_name_vector` are stored in `property_value_vector`.

The function `set` typically does not return a value. However, if called with an explicit return argument, for example, `a = set(target_object, property_name, property_value)`, it returns the values of the properties after the indicated settings have been made.

The properties for a scope object are listed in the following table. This table includes descriptions of the properties and the properties you can change directly by assigning a value.

Property	Description	Writable
`Application`	Name of the Simulink model associated with this scope object.	No
`Decimation`	A number `n`, where every `n`th sample is acquired in a scope window.	Yes
`NumPrePostSamples`	Number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set `TriggerMode` to `'FreeRun'`, this property has no effect on data acquisition.	Yes
`NumSamples`	Number of contiguous samples captured during the acquisition of a data package. If the scope stops before capturing this number of samples, the scope has the collected data up to the end of data collection, then has zeroes for the remaining uncollected data. Note that you should know what type of data you are collecting, it is possible that your data contains zeroes. For file scopes, this parameter works in conjunction with the **AutoRestart** check box. If the **AutoRestart** box is selected, the file scope collects data up to **Number of Samples**, then starts over again, overwriting the buffer. If the **AutoRestart** box is not selected, the file scope collects data only up to **Number of Samples**, then stops.	Yes
`ScopeId`	A numeric index, unique for each scope.	No
`Signals`	List of signal indices from the target object to display on the scope.	Yes

Property	Description	Writable
`Status`	Indicate whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are `'Acquiring'`, `'Ready for being Triggered'`, `'Interrupted'`, and `'Finished'`.	No
`TriggerLevel`	If `TriggerMode` is `'Signal'`, indicates the value the signal has to cross to trigger the scope and start acquiring data. The trigger level can be crossed with either a rising or falling signal.	Yes
`TriggerMode`	Trigger mode for a scope. Valid values are `'FreeRun'` (default), `'Software'`, `'Signal'`, and `'Scope'`.	Yes
`TriggerSample`	If `TriggerMode` is `'Scope'`, then `TriggerSample` specifies which sample of the triggering scope the current scope should trigger on. For example, if `TriggerSample` is `0` (default), the current scope triggers on sample `0` (first sample acquired) of the triggering scope. This means that the two scopes will be perfectly synchronized. If `TriggerSample` is `1`, the first sample (sample `0`) of the current scope will be at the same instant as sample number 1 (second sample in the acquisition cycle) of the triggering scope. As a special case, setting `TriggerSample` to `-1` means that the current scope is triggered at the end of the acquisition cycle of the triggering scope. Thus, the first sample of the triggering scope is acquired one sample after the last sample of the triggering scope.	Yes

Property	Description	Writable
TriggerScope	If TriggerMode is 'Scope', identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered. You do this by setting the slave scope property TriggerScope to the scope index of the master scope.	Yes
TriggerSignal	If TriggerMode is 'Signal', identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property Signal.	Yes
TriggerSlope	If TriggerMode is 'Signal', indicates whether the trigger is on a rising or falling signal. Values are 'Either' (default), 'Rising', and 'Falling'.	Yes
Type	Determines whether the scope is displayed on the host computer or on the target computer. Values are 'Host', 'Target', and 'File'. Property Type is set only once, when the scope is created on the target computer.	No

Examples

Get a list of writable properties for a scope object.

```
sc1 = getscope(tg,1)
set(sc1)
ans=
           NumSamples: {}
           Decimation: {}
          TriggerMode: {5x1 cell}
        TriggerSignal: {}
         TriggerLevel: {}
         TriggerSlope: {4x1 cell}
         TriggerScope: {}
        TriggerSample: {}
              Signals: {}
    NumPrePostSamples: {}
```

```
  Mode: {5x1 cell}
YLimit: {}
  Grid: {}
```

The property value for the scope object `sc1` is changed to `on`:

```
sc1.set('grid', 'on') or set(sc1, 'grid', 'on')
```

See Also

`set` | `SimulinkRealTime.fileScope.get` | `SimulinkRealTime.targetScope.get` | `SimulinkRealTime.target.set`

SimulinkRealTime.hostScope.start

Purpose

Start execution of scope on target computer

Syntax

MATLAB command line

```
start(scope_object_vector)
scope_object_vector.start
+scope_object_vector
start(getscope((target_object, signal_index_vector))
```

Target computer command line

```
startscope scope_index
startscope 'all'
```

Arguments

`target_object`	Name of a target object.
`scope_object_vector`	Name of a single scope object, name of vector of scope objects, list of scope object names in vector form `[scope_object1, scope_object2]`, or the target object method `getscope`, which returns a `scope_object` vector.
`signal_index_vector`	Index for a single scope or list of scope indices in vector form.
`scope_index`	Single scope index.

Description

Method for a scope object. Starts a scope on the target computer represented by a scope object on the host computer. This method might not start data acquisition, which depends on the trigger settings. Before using this method, you must create a scope. To create a scope, use the target object method `addscope` or add Simulink Real-Time scope blocks to your Simulink model.

Examples

Start one scope with the scope object sc1.

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

or type

```
start(getscope(tg,1))
```

Start two scopes.

```
somescopes = getscope(tg,[1,2]) or somescopes=
tg.getscope([1,2])
start(somescopes) or somescopes.start
```

or type

```
sc1 = getscope(tg,1) or sc1 =tg.getscope(1)
sc2 = getscope(tg,2) or sc2 = tg.getscope(2)
start([sc1,sc2])
```

or type

```
start(getscope(tg,[1,2])
```

Start all scopes:

```
allscopes = getscope(tg) or allscopes = tg.getscope
start(allscopes) or allscopes.start or +allscopes
```

or type

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

See Also

`SimulinkRealTime.fileScope.stop` |
`SimulinkRealTime.targetScope.stop` |
`SimulinkRealTime.target.getscope` |
`SimulinkRealTime.target.start`

SimulinkRealTime.hostScope.stop

Purpose

Stop execution of scope on target computer

Syntax

MATLAB command line

```
stop(scope_object_vector)
scope_object.stop
-scope_object
stop(getscope(target_object, signal_index_vector))
```

Target computer command line

```
stopscope scope_index
stopscope 'all'
```

Arguments

`target_object`	Name of a target object.
`scope_object_vector`	Name of a single scope object, name of vector of scope objects, list of scope object names in a vector form `[scope_object1, scope_object2]`, or the target object method `getscope`, which returns a `scope_object` vector.
`signal_index_vector`	Index for a single scope or list of scope indices in vector form.
`scope_index`	Single scope index.

Description

Method for scope objects. Stops the scopes represented by the scope objects.

Examples

Stop one scope represented by the scope object `sc1`.

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

Stop all scopes with a scope object vector `allscopes` created with the command

```
allscopes = getscope(tg) or allscopes = tg.getscope.
stop(allscopes) or allscopes.stop or -allscopes
```

or type

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

See Also

`SimulinkRealTime.fileScope.start` | `SimulinkRealTime.targetScope.start` | `SimulinkRealTime.target.getscope` | `SimulinkRealTime.target.stop`

SimulinkRealTime.hostScope.trigger

Purpose

Software-trigger start of data acquisition for scope(s)

Syntax

`trigger(scope_object_vector) or scope_object_vector.trigger`

Arguments

`scope_object_vector`	Name of a single scope object, name of a vector of scope objects, list of scope object names in a vector form `[scope_object1, scope_object2]`, or the target object method `getscope`, which returns a `scope_object` vector.

Description

Method for a scope object. If the scope object property `TriggerMode` has a value of `'software'`, this function triggers the scope represented by the scope object to acquire the number of data points in the scope object property `NumSamples`.

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

Examples

Set a single scope to software trigger, trigger the acquisition of one set of samples, and plot data.

```
sc1 = tg.addscope('host',1) or sc1=addscope(tg,'host',1)
sc1.triggermode = 'software'
tg.start, or start(tg), or +tg
sc1.start  or start(sc1) or +sc1
sc1.trigger or trigger(sc1)
plot(sc1.time, sc1.data)
sc1.stop or stop(sc1) or -sc1
tg.stop or stop(tg) or -tg1
```

Set all scopes to software trigger and trigger to start.

```
allscopes = tg.getscopes
allscopes.triggermode = 'software'
allscopes.start or start(allscopes) or +allscopes
allscopes.trigger or trigger(allscopes)
```

SimulinkRealTime.targetScope

Purpose

Control and access properties of target scopes

Description

The kernel acquires a data package and the scope displays the data on the target computer screen. Depending on the setting of `DisplayMode`, the data may be displayed numerically or graphically by a redrawing, sliding, and rolling display.

Methods

These methods are held in common by file, host, and target scopes.

Method	Description
`SimulinkRealTime.targetScope.addsignal`	Add signals to scope represented by scope object
`SimulinkRealTime.targetScope.get`	Return property values for scope object
`SimulinkRealTime.targetScope.remsignal`	Remove signals from scope represented by scope object
`SimulinkRealTime.targetScope.set`	Change property values for scope object
`SimulinkRealTime.targetScope.start`	Start execution of scope on target computer
`SimulinkRealTime.targetScope.stop`	Stop execution of scope on target computer
`SimulinkRealTime.targetScope.trigger`	Software trigger start of data acquisition for scope or scopes

Properties

These properties are held in common by file, host, and target scopes.

Property	Description	Writable
`Application`	Name of the Simulink model associated with this scope object.	No
`Decimation`	A number `n`, where every `n`th sample is acquired in a scope window.	Yes

Property	Description	Writable
`NumPrePostSamples`	Number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set `TriggerMode` to `'FreeRun'`, this property has no effect on data acquisition.	Yes
`NumSamples`	Number of contiguous samples captured during the acquisition of a data package. If the scope stops before capturing this number of samples, the scope has the collected data up to the end of data collection, then has zeroes for the remaining uncollected data. Note that you should know what type of data you are collecting, it is possible that your data contains zeroes. For file scopes, this parameter works in conjunction with the **AutoRestart** check box. If the **AutoRestart** box is selected, the file scope collects data up to **Number of Samples**, then starts over again, overwriting the buffer. If the **AutoRestart** box is not selected, the file scope collects data only up to **Number of Samples**, then stops.	Yes
`ScopeId`	A numeric index, unique for each scope.	No
`Signals`	List of signal indices from the target object to display on the scope.	Yes
`Status`	Indicate whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are `'Acquiring'`, `'Ready for being Triggered'`, `'Interrupted'`, and `'Finished'`.	No

Property	Description	Writable
`TriggerLevel`	If `TriggerMode` is `'Signal'`, indicates the value the signal has to cross to trigger the scope and start acquiring data. The trigger level can be crossed with either a rising or falling signal.	Yes
`TriggerMode`	Trigger mode for a scope. Valid values are `'FreeRun'` (default), `'Software'`, `'Signal'`, and `'Scope'`.	Yes
`TriggerSample`	If `TriggerMode` is `'Scope'`, then `TriggerSample` specifies which sample of the triggering scope the current scope should trigger on. For example, if `TriggerSample` is `0` (default), the current scope triggers on sample `0` (first sample acquired) of the triggering scope. This means that the two scopes will be perfectly synchronized. If `TriggerSample` is `1`, the first sample (sample `0`) of the current scope will be at the same instant as sample number 1 (second sample in the acquisition cycle) of the triggering scope. As a special case, setting `TriggerSample` to `-1` means that the current scope is triggered at the end of the acquisition cycle of the triggering scope. Thus, the first sample of the triggering scope is acquired one sample after the last sample of the triggering scope.	Yes
`TriggerScope`	If `TriggerMode` is `'Scope'`, identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered. You do this by setting the slave scope property `TriggerScope` to the scope index of the master scope.	Yes

Property	Description	Writable
`TriggerSignal`	If `TriggerMode` is `'Signal'`, identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property `Signal`.	Yes
`TriggerSlope`	If `TriggerMode` is `'Signal'`, indicates whether the trigger is on a rising or falling signal. Values are `'Either'` (default), `'Rising'`, and `'Falling'`.	Yes
`Type`	Determines whether the scope is displayed on the host computer or on the target computer. Values are `'Host'`, `'Target'`, and `'File'`. Property `Type` is set only once, when the scope is created on the target computer.	No

These properties are specific to target scopes.

Property	Description	Writeable
`DisplayMode`	For target scopes, indicate how a scope displays the signals. Values are `'Numerical'`, `'Redraw'` (default), `'Sliding'`, and `'Rolling'`. For host or file scopes, this parameter has no effect. .	Yes
`Grid`	Values are `'on'` and `'off'`. For host or file scopes, this parameter has no effect.	Yes

Property	Description	Writeable
`Mode`	**Note** The `Mode` property will be removed in a future release. • For target scopes, use `DisplayMode`. • For file scopes, use `WriteMode`. • For host scopes, this parameter has no effect.	Yes
`YLimit`	Minimum and maximum *y*-axis values. This property can be set to `'auto'`. For host or file scopes, this parameter has no effect.	Yes

SimulinkRealTime.targetScope.addsignal

Purpose

Add signals to scope represented by scope object

Syntax

MATLAB command line

```
addsignal(scope_object_vector, signal_index_vector)
scope_object_vector.addsignal(signal_index_vector)
```

Target command line

```
addsignal scope_index = signal_index, signal_index, . . .
```

Arguments

`scope_object_vector`	Name of a single scope object or the name of a vector of scope objects.
`signal_index_vector`	For one signal, use a single number. For two or more signals, enclose numbers in brackets and separate with commas.
`scope_index`	Single scope index.

Description

`addsignal` adds signals to a scope object. The signals must be specified by their indices, which you can retrieve using the target object method `getsignalid`. If the `scope_object_vector` has two or more scope objects, the same signals are assigned to each scope.

Note You must stop the scope before you can add a signal to it.

Examples

Add signals `0` and `1` from the target object `tg` to the scope object `sc1`. The signals are added to the scope, and the scope object property `Signals` is updated to include the added signals.

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

Display a list of properties and values for the scope object `sc1` with the property `Signals`, as shown below.

```
sc1.Signals
Signals                 = 1  : Signal Generator
                          0  : Integrator1
```

Another way to add signals without using the method `addsignal` is to use the scope object method `set`.

```
set(sc1,'Signals', [0,1]) or sc1.set('signals',[0,1]
```

Or, to directly assign signal values to the scope object property `Signals`,

```
sc1.signals = [0,1]
```

See Also

`SimulinkRealTime.fileScope.remsignal`
`| SimulinkRealTime.fileScope.set |`
`SimulinkRealTime.target.getsignalid`

How To

- “Target Scope Usage”
- “Host Scope Usage”
- “File Scope Usage”
- “Application and Driver Scripts”

Purpose

Return property values for scope objects

Syntax

```
get(scope_object_vector)
get(scope_object_vector, 'scope_object_property')
get(scope_object_vector, scope_object_property_vector)
```

Arguments

`target_object`	Name of a target object.
`scope_object_vector`	Name of a single scope or name of a vector of scope objects.
`scope_object_property`	Name of a scope object property.

Description

`get` gets the value of readable scope object properties from a scope object or the same property from each scope object in a vector of scope objects. Scope object properties let you select signals to acquire, set triggering modes, and access signal information from the target application. You can view and change these properties using scope object methods.

The properties for a scope object are listed in the following table. This table includes descriptions of the properties and the properties you can change directly by assigning a value.

Property	Description	Writable
`Application`	Name of the Simulink model associated with this scope object.	No
`Decimation`	A number `n`, where every `n`th sample is acquired in a scope window.	Yes
`NumPrePostSamples`	Number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set `TriggerMode` to `'FreeRun'`, this property has no effect on data acquisition.	Yes

Property	Description	Writable
`NumSamples`	Number of contiguous samples captured during the acquisition of a data package. If the scope stops before capturing this number of samples, the scope has the collected data up to the end of data collection, then has zeroes for the remaining uncollected data. Note that you should know what type of data you are collecting, it is possible that your data contains zeroes. For file scopes, this parameter works in conjunction with the **AutoRestart** check box. If the **AutoRestart** box is selected, the file scope collects data up to **Number of Samples**, then starts over again, overwriting the buffer. If the **AutoRestart** box is not selected, the file scope collects data only up to **Number of Samples**, then stops.	Yes
`ScopeId`	A numeric index, unique for each scope.	No
`Signals`	List of signal indices from the target object to display on the scope.	Yes
`Status`	Indicate whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are `'Acquiring'`, `'Ready for being Triggered'`, `'Interrupted'`, and `'Finished'`.	No
`TriggerLevel`	If `TriggerMode` is `'Signal'`, indicates the value the signal has to cross to trigger the scope and start acquiring data. The trigger level can be crossed with either a rising or falling signal.	Yes
`TriggerMode`	Trigger mode for a scope. Valid values are `'FreeRun'` (default), `'Software'`, `'Signal'`, and `'Scope'`.	Yes

Property	Description	Writable
`TriggerSample`	If `TriggerMode` is `'Scope'`, then `TriggerSample` specifies which sample of the triggering scope the current scope should trigger on. For example, if `TriggerSample` is `0` (default), the current scope triggers on sample `0` (first sample acquired) of the triggering scope. This means that the two scopes will be perfectly synchronized. If `TriggerSample` is `1`, the first sample (sample `0`) of the current scope will be at the same instant as sample number 1 (second sample in the acquisition cycle) of the triggering scope. As a special case, setting `TriggerSample` to `-1` means that the current scope is triggered at the end of the acquisition cycle of the triggering scope. Thus, the first sample of the triggering scope is acquired one sample after the last sample of the triggering scope.	Yes
`TriggerScope`	If `TriggerMode` is `'Scope'`, identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered. You do this by setting the slave scope property `TriggerScope` to the scope index of the master scope.	Yes
`TriggerSignal`	If `TriggerMode` is `'Signal'`, identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property `Signal`.	Yes

Property	Description	Writable
`TriggerSlope`	If `TriggerMode` is `'Signal'`, indicates whether the trigger is on a rising or falling signal. Values are `'Either'` (default), `'Rising'`, and `'Falling'`.	Yes
`Type`	Determines whether the scope is displayed on the host computer or on the target computer. Values are `'Host'`, `'Target'`, and `'File'`. Property `Type` is set only once, when the scope is created on the target computer.	No

Examples

List the readable properties, along with their current values. This is given in the form of a structure whose field names are the property names and whose field values are property values.

```
get(sc)
```

List the value for the scope object property `Type`. Notice that the property name is a string, in quotation marks, and is not case sensitive.

```
get(sc,'type')
ans = Target
```

See Also

`SimulinkRealTime.fileScope.set` | `SimulinkRealTime.hostScope.set` | `get` | `SimulinkRealTime.target.get`

SimulinkRealTime.targetScope.remsignal

Purpose

Remove signals from scope represented by scope object

Syntax

MATLAB command line

```
remsignal(scope_object)
remsignal(scope_object, signal_index_vector)
scope_object.remsignal(signal_index_vector)
```

Target command line

```
remsignal scope_index = signal_index, signal_index, . . .
```

Arguments

`scope_object`	MATLAB object created with the target object method `addscope` or `getscope`.
`signal_index_vector`	Index numbers from the scope object property `Signals`. This argument is optional, and if it is left out all signals are removed.
`signal_index`	Single signal index.

Description

`remsignal` removes signals from a scope object. The signals must be specified by their indices, which you can retrieve using the target object method `getsignalid`. If the `scope_index_vector` has two or more scope objects, the same signals are removed from each scope. The argument `signal_index` is optional; if it is left out, all signals are removed.

Note You must stop the scope before you can remove a signal from it.

Examples

Remove signals 0 and 1 from the scope represented by the scope object sc1.

```
sc1.get('signals')
ans= 0 1
```

Remove signals from the scope on the target computer with the scope object property `Signals` updated.

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

or

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

See Also

`SimulinkRealTime.hostScope.addsignal` |
`SimulinkRealTime.targetScope.addsignal` |
`SimulinkRealTime.target.getsignalid`

Purpose

Change property values for scope objects

Syntax

```
set(scope_object_vector)
set(scope_object_vector, property_name1, property_value1,
property_name2, property_value2, . . .)
scope_object_vector.set('property_name1', property_value1,
. . .)
set(scope_object, 'property_name', property_value, . . .)
```

Arguments

`scope_object`	Name of a scope object or a vector of scope objects.
`'property_name'`	Name of a scope object property. Always use quotation marks.
`property_value`	Value for a scope object property. Always use quotation marks for character strings; quotation marks are optional for numbers.

Description

Method for scope objects. Sets the properties of the scope object. Not all properties are user writable. Scope object properties let you select signals to acquire, set triggering modes, and access signal information from the target application. You can view and change these properties using scope object methods.

Properties must be entered in pairs or, using the alternate syntax, as one-dimensional cell arrays of the same size. This means they must both be row vectors or both column vectors, and the corresponding values for properties in `property_name_vector` are stored in `property_value_vector`.

The function `set` typically does not return a value. However, if called with an explicit return argument, for example, `a = set(target_object, property_name, property_value)`, it returns the values of the properties after the indicated settings have been made.

The properties for a scope object are listed in the following table. This table includes descriptions of the properties and the properties you can change directly by assigning a value.

Property	Description	Writable
`Application`	Name of the Simulink model associated with this scope object.	No
`Decimation`	A number `n`, where every `n`th sample is acquired in a scope window.	Yes
`NumPrePostSamples`	Number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set `TriggerMode` to `'FreeRun'`, this property has no effect on data acquisition.	Yes
`NumSamples`	Number of contiguous samples captured during the acquisition of a data package. If the scope stops before capturing this number of samples, the scope has the collected data up to the end of data collection, then has zeroes for the remaining uncollected data. Note that you should know what type of data you are collecting, it is possible that your data contains zeroes. For file scopes, this parameter works in conjunction with the **AutoRestart** check box. If the **AutoRestart** box is selected, the file scope collects data up to **Number of Samples**, then starts over again, overwriting the buffer. If the **AutoRestart** box is not selected, the file scope collects data only up to **Number of Samples**, then stops.	Yes
`ScopeId`	A numeric index, unique for each scope.	No
`Signals`	List of signal indices from the target object to display on the scope.	Yes

Property	Description	Writable
`Status`	Indicate whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are `'Acquiring'`, `'Ready for being Triggered'`, `'Interrupted'`, and `'Finished'`.	No
`TriggerLevel`	If `TriggerMode` is `'Signal'`, indicates the value the signal has to cross to trigger the scope and start acquiring data. The trigger level can be crossed with either a rising or falling signal.	Yes
`TriggerMode`	Trigger mode for a scope. Valid values are `'FreeRun'` (default), `'Software'`, `'Signal'`, and `'Scope'`.	Yes
`TriggerSample`	If `TriggerMode` is `'Scope'`, then `TriggerSample` specifies which sample of the triggering scope the current scope should trigger on. For example, if `TriggerSample` is `0` (default), the current scope triggers on sample `0` (first sample acquired) of the triggering scope. This means that the two scopes will be perfectly synchronized. If `TriggerSample` is `1`, the first sample (sample `0`) of the current scope will be at the same instant as sample number 1 (second sample in the acquisition cycle) of the triggering scope. As a special case, setting `TriggerSample` to `-1` means that the current scope is triggered at the end of the acquisition cycle of the triggering scope. Thus, the first sample of the triggering scope is acquired one sample after the last sample of the triggering scope.	Yes

Property	Description	Writable
`TriggerScope`	If `TriggerMode` is `'Scope'`, identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered. You do this by setting the slave scope property `TriggerScope` to the scope index of the master scope.	Yes
`TriggerSignal`	If `TriggerMode` is `'Signal'`, identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property `Signal`.	Yes
`TriggerSlope`	If `TriggerMode` is `'Signal'`, indicates whether the trigger is on a rising or falling signal. Values are `'Either'` (default), `'Rising'`, and `'Falling'`.	Yes
`Type`	Determines whether the scope is displayed on the host computer or on the target computer. Values are `'Host'`, `'Target'`, and `'File'`. Property `Type` is set only once, when the scope is created on the target computer.	No

Examples

Get a list of writable properties for a scope object.

```
sc1 = getscope(tg,1)
set(sc1)
ans=
           NumSamples: {}
           Decimation: {}
          TriggerMode: {5x1 cell}
        TriggerSignal: {}
         TriggerLevel: {}
         TriggerSlope: {4x1 cell}
         TriggerScope: {}
        TriggerSample: {}
              Signals: {}
    NumPrePostSamples: {}
```

```
      Mode: {5x1 cell}
    YLimit: {}
      Grid: {}
```

The property value for the scope object `sc1` is changed to `on`:

```
sc1.set('grid', 'on') or set(sc1, 'grid', 'on')
```

See Also

`set` | `SimulinkRealTime.hostScope.get` | `SimulinkRealTime.targetScope.get` | `SimulinkRealTime.target.set`

SimulinkRealTime.targetScope.start

Purpose

Start execution of scope on target computer

Syntax

MATLAB command line

```
start(scope_object_vector)
scope_object_vector.start
+scope_object_vector
start(getscope((target_object, signal_index_vector))
```

Target computer command line

```
startscope scope_index
startscope 'all'
```

Arguments

`target_object`	Name of a target object.
`scope_object_vector`	Name of a single scope object, name of vector of scope objects, list of scope object names in vector form `[scope_object1, scope_object2]`, or the target object method `getscope`, which returns a `scope_object` vector.
`signal_index_vector`	Index for a single scope or list of scope indices in vector form.
`scope_index`	Single scope index.

Description

Method for a scope object. Starts a scope on the target computer represented by a scope object on the host computer. This method might not start data acquisition, which depends on the trigger settings. Before using this method, you must create a scope. To create a scope, use the target object method `addscope` or add Simulink Real-Time scope blocks to your Simulink model.

Examples

Start one scope with the scope object sc1.

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

or type

```
start(getscope(tg,1))
```

Start two scopes.

```
somescopes = getscope(tg,[1,2]) or somescopes=
tg.getscope([1,2])
start(somescopes) or somescopes.start
```

or type

```
sc1 = getscope(tg,1) or sc1 =tg.getscope(1)
sc2 = getscope(tg,2) or sc2 = tg.getscope(2)
start([sc1,sc2])
```

or type

```
start(getscope(tg,[1,2])
```

Start all scopes:

```
allscopes = getscope(tg) or allscopes = tg.getscope
start(allscopes) or allscopes.start or +allscopes
```

or type

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

See Also

`SimulinkRealTime.hostScope.stop` |
`SimulinkRealTime.targetScope.stop` |
`SimulinkRealTime.target.getscope` |
`SimulinkRealTime.target.start`

SimulinkRealTime.targetScope.stop

Purpose

Stop execution of scope on target computer

Syntax

MATLAB command line

```
stop(scope_object_vector)
scope_object.stop
-scope_object
stop(getscope(target_object, signal_index_vector))
```

Target computer command line

```
stopscope scope_index
stopscope 'all'
```

Arguments

`target_object`	Name of a target object.
`scope_object_vector`	Name of a single scope object, name of vector of scope objects, list of scope object names in a vector form `[scope_object1, scope_object2]`, or the target object method `getscope`, which returns a `scope_object` vector.
`signal_index_vector`	Index for a single scope or list of scope indices in vector form.
`scope_index`	Single scope index.

Description

Method for scope objects. Stops the scopes represented by the scope objects.

Examples

Stop one scope represented by the scope object `sc1`.

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

Stop all scopes with a scope object vector `allscopes` created with the command

```
allscopes = getscope(tg) or allscopes = tg.getscope.
stop(allscopes) or allscopes.stop or -allscopes
```

or type

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

See Also

`SimulinkRealTime.hostScope.start` | `SimulinkRealTime.targetScope.start` | `SimulinkRealTime.target.getscope` | `SimulinkRealTime.target.stop`

SimulinkRealTime.targetScope.trigger

Purpose

Software-trigger start of data acquisition for scope(s)

Syntax

`trigger(scope_object_vector) or scope_object_vector.trigger`

Arguments

`scope_object_vector`	Name of a single scope object, name of a vector of scope objects, list of scope object names in a vector form `[scope_object1, scope_object2]`, or the target object method `getscope`, which returns a `scope_object` vector.

Description

Method for a scope object. If the scope object property `TriggerMode` has a value of `'software'`, this function triggers the scope represented by the scope object to acquire the number of data points in the scope object property `NumSamples`.

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

Examples

Set a single scope to software trigger, trigger the acquisition of one set of samples, and plot data.

```
sc1 = tg.addscope('host',1) or sc1=addscope(tg,'host',1)
sc1.triggermode = 'software'
tg.start, or start(tg), or +tg
sc1.start  or start(sc1) or +sc1
sc1.trigger or trigger(sc1)
plot(sc1.time, sc1.data)
sc1.stop or stop(sc1) or -sc1
tg.stop or stop(tg) or -tg1
```

Set all scopes to software trigger and trigger to start.

```
allscopes = tg.getscopes
allscopes.triggermode = 'software'
allscopes.start or start(allscopes) or +allscopes
allscopes.trigger or trigger(allscopes)
```