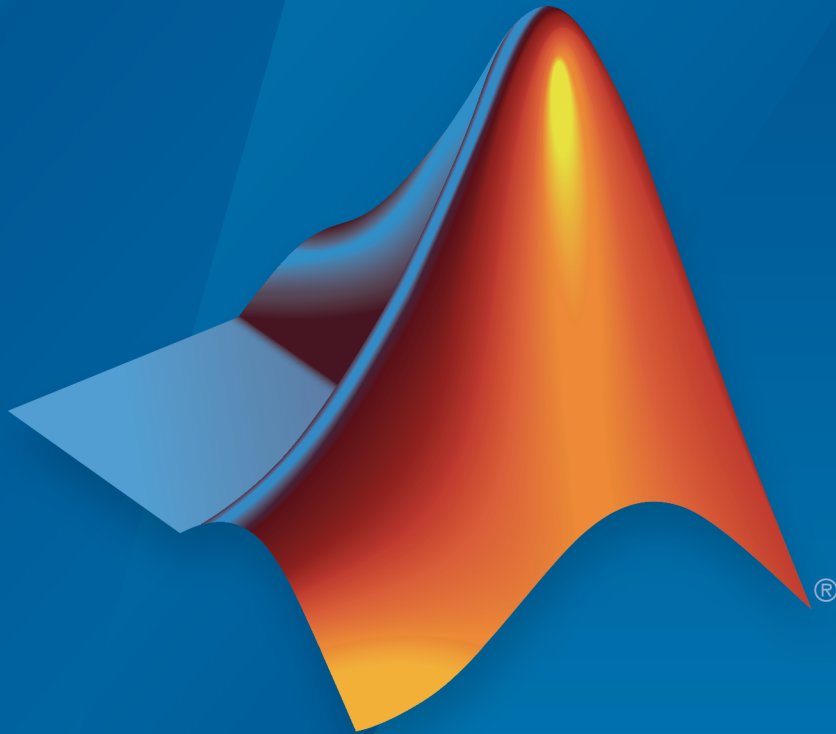


Simulink® Real-Time™

API Guide



MATLAB® & SIMULINK®

R2016b



How to Contact MathWorks



Latest news: www.mathworks.com
Sales and services: www.mathworks.com/sales_and_services
User community: www.mathworks.com/matlabcentral
Technical support: www.mathworks.com/support/contact_us



Phone: 508-647-7000



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

Simulink[®] Real-Time[™] API Guide

© COPYRIGHT 2002–2016 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)
October 2014	Online only	Revised for Version 6.1 (Release 2014b)
March 2015	Online only	Revised for Version 6.2 (Release 2015a)
September 2015	Online only	Revised for Version 6.3 (Release 2015b)
March 2016	Online only	Revised for Version 6.4 (Release 2016a)
September 2016	Online only	Revised for Version 6.5 (Release 2016b)

Introduction

1

Simulink Real-Time API for Microsoft .NET Framework . . .	1-2
xPCTargetPC Class	1-4
xPCApplication Class	1-5
xPCFileSystem	1-5
Simulink Real-Time C API	1-7
C API Error Messages	1-8

Simulink Real-Time API for Microsoft .NET Framework

2

Using the Simulink Real-Time API for Microsoft .NET Framework	2-2
Simulink Real-Time .NET API Application Creation	2-4
Visual Studio Coding Environment	2-4
Visual Studio Design Environment	2-5
Simulink Real-Time .NET API Application Distribution . . .	2-6
Simulink Real-Time .NET API Client Application Examples	2-7

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

3

Simulink Real-Time API for C

4

Using the C API 4-2

Simulink Real-Time API Reference for C

5

MATLAB API

6

Introduction

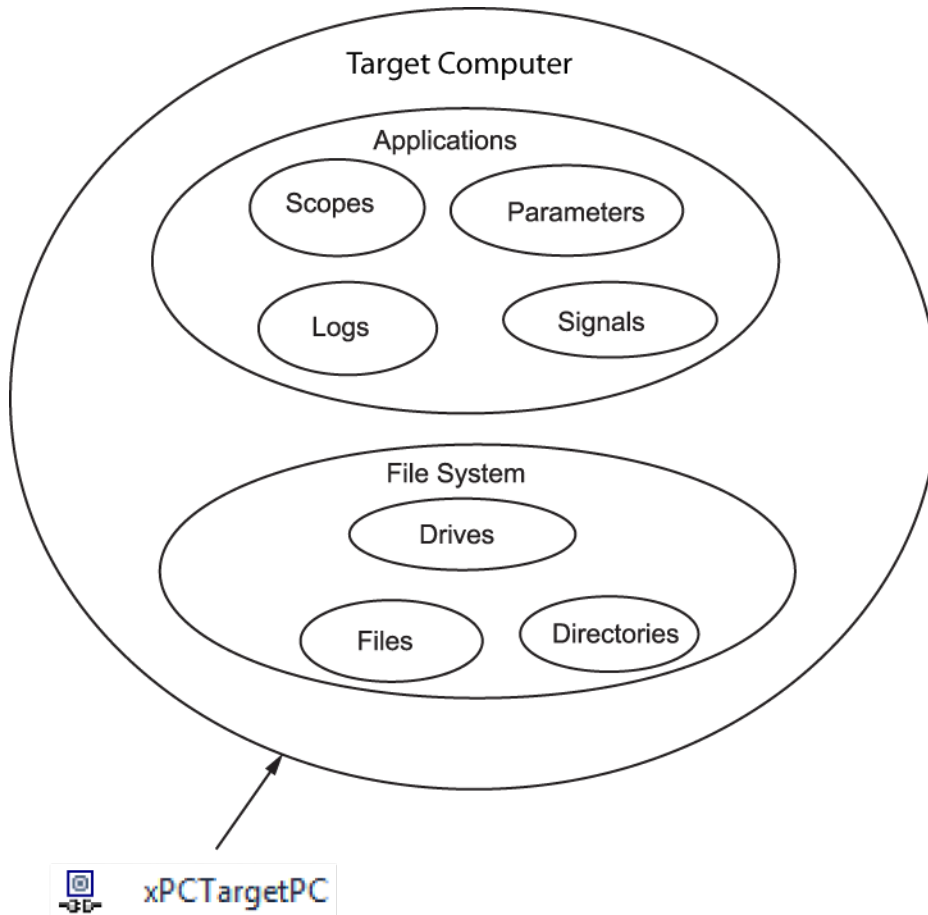
- “Simulink Real-Time API for Microsoft .NET Framework” on page 1-2
- “Simulink Real-Time C API” on page 1-7
- “C API Error Messages” on page 1-8

Simulink Real-Time API for Microsoft .NET Framework

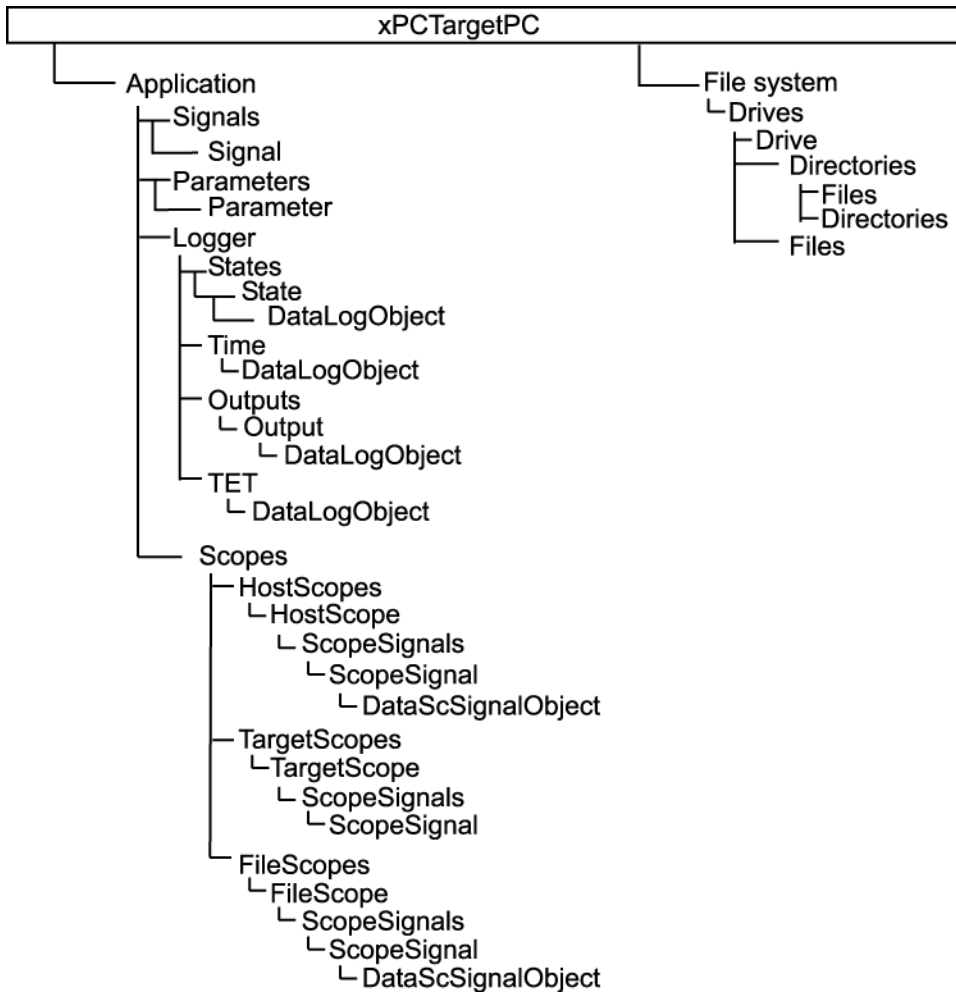
The Simulink Real-Time API for Microsoft .NET Framework consists of objects arranged in hierarchical order. Each of these objects has functions and properties that allow you to manipulate and interact with the API. The API provides a number of object types, including those for the target computer, real-time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

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

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



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



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

xPCTargetPC Class

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

The xPCTargetPC object is at the root level of the object model. After you connect the .NET application running on the development computer to the real-time application

running on the target computer, the object exposes information about the Simulink Real-Time session. `xPCTargetPC` provides many member functions that you use to access information and to manipulate the real-time application and the target computer file system.

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

xPCApplication Class

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

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

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

xPCFileSystem

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

An `xPCFileSystem` object contains objects like the following:

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

Simulink Real-Time C API

The Simulink Real-Time C API consists of a series of C functions that you can call from a C or C++ custom program. This API is designed for multi-threaded operation on a 64-bit target computer.

The Simulink Real-Time C API DLL consists of C functions that you can incorporate into a custom program. A user can use an application written through either interface to load, run, and monitor a real-time application without interacting with MATLAB. Using the Simulink Real-Time C API, you write the custom program in a high-level language (such as C, C++, or Java[®]) that works with a real-time application. This option requires that you are an experienced programmer.

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

All custom Simulink Real-Time C API programs 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 program.

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

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

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

C API Error Messages

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

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

Message	Description
EINVNUMSIGNALS	Invalid number of signals
EINVPARIDX	Invalid parameter index
EINVPORT	Invalid Port Number
EINVSCIDX	Invalid Scope Index
EINVSTYPE	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)
EINWVSVER	Invalid Winsock version (1.1 needed)
EINXPCVERSION	Target has an invalid version of Simulink Real-Time
ELOADAPPFIRST	Load the application first
ELOGGINGDISABLED	Logging is disabled
EMALFORMED	Malformed message
EMEMALLOC	Memory allocation error
ENODATALOGGED	No data has been logged
ENOERR	No error
ENOFREEPORT	No free Port in C API
ENOMORECHANNELS	No more channels in scope
ENOSPACE	Space not allocated
EOUTPUTLOGDISABLED	Output Logging is disabled
EPARNOTFOUND	Parameter not found
EPARSIZMISMATCH	Parameter Size mismatch

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

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

Simulink Real-Time API for Microsoft .NET Framework

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

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

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

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

The Simulink Real-Time API for .NET framework provides multiple ways for you to interface client-side custom 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 (such as PowerShell) or as a process console standalone application (.exe file).

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

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

The Simulink Real-Time API for .NET Framework does not support applications that use the .NET client profile. It only supports applications that use the full .NET Framework.

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

For some examples of custom .NET applications, see “Simulink Real-Time .NET API Client Application Examples” on page 2-7

Simulink Real-Time .NET API Application Creation

Before creating your Microsoft .NET Framework custom client application, set up the development environment. In addition to installing the products listed in the system requirements at www.mathworks.com/products/xpctarget/requirements.html, do the following setup.

Visual Studio Coding 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, such as Microsoft Visual Studio.
- To build an application (.exe or DLL) that calls functions from the Simulink Real-Time API libraries, use a third-party compiler that generates code for Win64 computers. You can write client applications that call these functions in another high-level language, such as C#, C++, or C.
- Create a Windows application.
- To run the application on a 64-bit computer, copy `xpcapi.dll` file from `matlabroot\toolbox\rtw\targets\xpc\api\x64` to the folder where you build the executable application.
- Add a reference for `xPCFramework.dll` to your project by including the following in your code.

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

You can then access the types available from the Simulink Real-Time environment, for example, when creating a console or graphic display application.

- Compile your Microsoft .NET Framework client application as a 64-bit application.

You can connect a target computer to only one development computer at a time. Before starting your .NET application, be sure to disconnect the target computer from the development computer (`xPCTargetPC.disconnect`). You can use the `slrtpingtarget` from the Command Window to verify that the development and target computers are not connected. When execution is finished, this function disconnects from the target computer.

If your development computer has additional network resources, you can connect additional target computers to the same development computer.

When your .NET application starts, first connect the development computer to the target computer (`xPCTargetPC.connect`), and then test the link between the development and target computers (`xPCTargetPC.ping`).

Visual Studio Design Environment

Optionally, you can use the design-time capability of the Microsoft Visual Studio environment with the `xPCTargetPC` nonvisual component. To make these capabilities available, carry out the following steps.

- 1 Add `xPCFramework.dll` to the Visual Studio Toolbox.
- 2 Add an `xPCTargetPC` object to the application form by dragging an `xPCTargetPC` control from the Toolbox window to the design surface.
- 3 To explore and customize the `xPCTargetPC` properties, click the `xPCTargetPC` control in the design surface.

The Visual Studio **Properties** window opens. In the **Properties** window, the `xPCTargetPC` control makes available its data and appearance properties.

Simulink Real-Time .NET API Application Distribution

To distribute your Microsoft .NET Framework client application, such as a user interface:

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

Simulink Real-Time .NET API Client Application Examples

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

The example “Simple Client Application With the .NET API” shows two client applications, **Example 1** and **Example 2**.

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

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

`FileSystemBrowse` is located in:

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

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

Simulink Real-Time API Reference for Microsoft .NET Framework

xPCFileScopeCollection.Add

Create xPCFileScope object with next available scope ID as key

Syntax

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

Description

Class: xPCFileScopeCollection Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

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

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

xPCFileScopeSignalCollection.Add

Add signals to file scope

Syntax

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

Description

Class: xPCFileScopeSignalCollection Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

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

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

`public IList<xPCFileScopeSignal> Add(int[] sigIds)` adds signals to the file scope. It creates an IList of xPCFileScopeSignal objects, one for each signal in the array

of IDs. *sigIds* is an array of 32-bit integers that specifies an array of IDs that represent the actual signals. This method returns an IList of xPCFileScopeSignal objects.

Exception

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

xPCHostScopeCollection.Add

Create xPCHostScope object with next available scope ID as key

Syntax

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

Description

Class: xPCHostScopeCollection Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

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

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

Exception

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

xPCHostScopeSignalCollection.Add

Add signals to host scope

Syntax

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

Description

Class: xPCHostScopeSignalCollection Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

`public xPCHostScopeSignal Add(string blkpath)` adds signal to the host scope. It creates an xPCHostScopeSignal object that *blkPath* specifies. *blkPath* is a character 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 ILLIST of xPCHostScopeSignal objects.

Exception

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

xPCTargetScopeCollection.Add

Create xPCTargetScope object

Syntax

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

Description

Class: xPCTargetScopeCollection Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

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

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

xPCTargetScopeSignalCollection.Add

Create xPCTargetScopeSignal object

Syntax

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

Description

Class: xPCTargetScopeSignalCollection Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

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

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

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

Exception

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

xPCFileStream.Close

Close current stream

Syntax

```
public void Close()
```

Description

Class: xPCFileStream Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

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

xPCTargetPC.Connect

Establish connection to target computer

Syntax

```
public void Connect()
```

Description

Class: xPCTargetPC Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

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

xPCTargetPC.ConnectAsync

Asynchronous request for target computer connection

Syntax

```
public void ConnectAsync()
```

Description

Class: xPCTargetPC Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

Exception	Condition
InvalidOperationException	When another thread uses this method.

xPCTargetPC.ConnectCompleted

Event when xPCTargetPC.ConnectAsync is complete

Syntax

```
public event ConnectCompleted ConnectCompleted
```

Description

Class: xPCTargetPC Class

Event

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCTargetPC.Connected

Event after xPCTargetPC.Connect is complete

Syntax

```
public event EventHandler Connected
```

Description

Class: xPCTargetPC Class

Event

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCTargetPC.Connecting

Event before xPCTargetPC.Connect starts

Syntax

```
public event EventHandler Connecting
```

Description

Class: xPCTargetPC Class

Event

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

public event EventHandler Connecting occurs before connect operation starts.

xPCFileInfo.CopyToHost

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

Syntax

```
public FileInfo CopyToHost(string DevelDestFileName)
```

Description

Class: xPCFileInfo Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

Exception	Condition
ArgumentException	<i>DevelDestFileName</i> is empty, contains only white spaces, or contains invalid characters.
ArgumentNullException	<i>DevelDestFileName</i> is NULL reference.
NotSupportedException	<i>DevelDestFileName</i> contains a colon (:) in the middle of the character string.
PathTooLongException	The specified path, file name, or both in <i>DevelDestFileName</i> 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 <i>DevelDestFileName</i> .
xPCException	When problem occurs, query xPCException object Reason property.

xPCFileInfo.Create

Create file in specified path

Syntax

```
public xPCFileStream Create()
```

Description

Class: xPCFileInfo Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

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

xPCFileSystem.CreateDirectory

Create folder

Syntax

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

Description

Class: xPCFileSystem Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

A fully qualified folder name can have a maximum of 248 characters, including the drive letter, colon, and backslash.

Exception

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

xPCDirectoryInfo.Create

Create folder

Syntax

```
public void Create()
```

Description

Class: xPCDirectoryInfo Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

`public void Create()` creates a folder.

xPCFileSystemInfo.Delete

Delete current file or folder

Syntax

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

Description

Class: xPCFileSystemInfo Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCDirectoryInfo.Delete

Delete empty xPCDirectoryInfo object

Syntax

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

Description

Class: xPCDirectoryInfo Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCFileInfo.Delete

Permanently delete file on target computer

Syntax

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

Description

Class: xPCFileInfo Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

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

xPCTargetPC.Disconnect

Disconnect from target computer

Syntax

```
public void Disconnect()
```

Description

Class: xPCTargetPC Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

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

xPCTargetPC.DisconnectAsync

Asynchronous request to disconnect from target computer

Syntax

```
public void DisconnectAsync()
```

Description

Class: xPCTargetPC Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

Exception	Condition
InvalidOperationException	When another thread uses this method.

xPCTargetPC.DisconnectCompleted

Event when xPCTargetPC.DisconnectAsync is complete

Syntax

```
public event DisconnectCompletedEventHandler DisconnectCompleted
```

Description

Class: xPCTargetPC Class

Event

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

```
public event DisconnectCompletedEventHandler DisconnectCompleted
```

occurs when an asynchronous disconnect operation is complete.

xPCTargetPC.Disconnected

Event after xPCTargetPC.Disconnect is complete

Syntax

```
public event EventHandler Disconnected
```

Description

Class: xPCTargetPC Class

Event

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCTargetPC.Disconnecting

Event before xPCTargetPC.Disconnect starts

Syntax

```
public event EventHandler Disconnecting
```

Description

Class: xPCTargetPC Class

Event

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

public event EventHandler Disconnecting occurs before a disconnect operation starts.

xPCTargetPC.Dispose

Clean up used resources

Syntax

```
public void Dispose()
```

Description

Class: xPCTargetPC Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

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

xPCTargetPC.Disposed

Event after xPCTargetPC.Dispose is complete

Syntax

```
public event EventHandler Disposed
```

Description

Class: xPCTargetPC Class

Event

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCFileSystem.GetCurrentDirectory

Current working folder for real-time application

Syntax

```
public string GetCurrentDirectory()
```

Description

Class: xPCFileSystem Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

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

xPCDataLoggingObject.GetData

Copy signal data from target computer

Syntax

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

Description

Class: xPCDataLoggingObject Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCDataFileScSignalObject.GetData

Copy file scope signal data from target computer

Syntax

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

Description

Class: xPCDataFileScSignalObject Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCDataHostScSignalObject.GetData

Copy host scope signal data from target computer

Syntax

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

Description

Class: xPCDataHostScSignalObject Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCDataLoggingObject.GetDataAsync

Asynchronously copy signal data from target computer

Syntax

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

Description

Class: xPCDataLoggingObject Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

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

xPCDataFileScSignalObject.GetDataAsync

Asynchronously copy file scope signal data from target computer

Syntax

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

Description

Class: xPCDataFileScSignalObject Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

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

Exception

Exception	Condition
InvalidOperationException	When another thread uses this method.

xPCDataHostScSignalObject.GetDataAsync

Asynchronously copy host scope signal data from target computer

Syntax

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

Description

Class: xPCDataHostScSignalObject Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

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

Exception

Exception	Condition
InvalidOperationException	When another thread uses this method.

xPCDataLoggingObject.GetDataCompleted

Event when xPCDataLoggingObject.GetDataAsync is complete

Syntax

```
public event GetDataCompletedEventHandler GetDataCompleted
```

Description

Class: xPCDataLoggingObject Class

Event

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCDataFileScSignalObject.GetDataCompleted

Event when xPCDataFileScSignalObject.GetDataAsync is complete

Syntax

```
public event GetFileScSignalDataCompletedEventHandler  
GetDataCompleted
```

Description

Class: xPCDataFileScSignalObject Class

Event

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

```
public event GetFileScSignalDataCompletedEventHandler  
GetDataCompleted
```

occurs when the asynchronous copying of file scope signal logged data is complete.

xPCDataHostScSignalObject.GetDataCompleted

Event when xPCDataHostScSignalObject.GetDataAsync is complete

Syntax

```
public event GetDataCompletedEventHandler GetDataCompleted
```

Description

Class: xPCDataHostScSignalObject Class

Event

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCDirectoryInfo.GetDirectories

Subfolders of current folder

Syntax

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

Description

Class: xPCDirectoryInfo Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCFileSystem.GetDrives

Drive names for logical drives on target computer

Syntax

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

Description

Class: xPCFileSystem Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

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

xPCDirectoryInfo.GetFiles

File list from current folder

Syntax

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

Description

Class: xPCDirectoryInfo Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCDirectoryInfo.GetFileSystemInfos

File system information for files and subfolders in folder

Syntax

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

Description

Class: xPCDirectoryInfo Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCParameter.GetParam

Get parameter values from target computer

Syntax

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

Description

Class: xPCParameter Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

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

xPCParameter.GetParamAsync

Asynchronous request to get parameter values from target computer

Syntax

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

Description

Class: xPCParameter Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

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

Exception

Exception	Condition
InvalidOperationException	When another thread uses this method.

xPCParameter.GetParamCompleted

Event when `xPCParameter.GetParamAsync` is complete

Description

Class: `xPCParameter` Class

Event

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: `C#`

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

xPCSignals.GetSignals

List of xPCSignal objects specified by array of signal identifiers

Syntax

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

Description

Class: xPCSignals Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

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

Exception

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

xPCSignals.GetSignalsValue

Vector of signal values from array

Syntax

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

Description

Class: xPCSignals Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

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

Exception

Exception	Condition
xPCException	When problem occurs, query <code>xPCException</code> object <code>Reason</code> property.

xPCSignal.GetValue

Value of signal at moment of request

Syntax

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

Description

Class: xPCSignal Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

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

xPCTargetPC.Load

Load real-time application onto target computer

Syntax

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

Description

Class: xPCTargetPC Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

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

Exception

Exception	Condition
ArgumentException	<i>DLMFileName</i> is empty, contains only white spaces, or contains invalid characters.
xPCException	When problem occurs, query <code>xPCException</code> object <code>Reason</code> property.

Exception	Condition
InvalidOperation-Exception	<i>DLMFileName</i> is a NULL reference (empty in Visual Basic®) or an empty character string.
NotSupportedException	<i>DLMFileName</i> contains a colon (:) in the middle of the character string.
PathTooLongException	The specified path, file name, or both in <i>DLMFileName</i> 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 <i>DLMFileName</i> .

xPCTargetPC.LoadAsync

Asynchronous request to load real-time application onto target computer

Syntax

```
public void LoadAsync()
```

Description

Class: xPCTargetPC Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

Exception	Condition
InvalidOperationException	When another thread uses this method.

xPCTargetPC.LoadCompleted

Event when xPCTargetPC.LoadAsync is complete

Syntax

```
public event LoadCompletedEventHandler LoadCompleted
```

Description

Class: xPCTargetPC Class

Event

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCTargetPC.Loaded

Event after xPCTargetPC.Load is complete

Syntax

```
public event EventHandler Loaded
```

Description

Class: xPCTargetPC Class

Event

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCTargetPC.Loading

Event before xPCTargetPC.Load starts

Syntax

```
public event EventHandler Loading
```

Description

Class: xPCTargetPC Class

Event

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCParameters.LoadParameterSet

Load parameter values for real-time application

Syntax

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

Description

Class: xPCParameters Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

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

CancelPropertyNotificationEventArgs Class

CancelPropertyNotification event data

Syntax

```
public class CancelPropertyNotificationEventArgs :  
PropertyNotificationEventArgs
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Properties

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

ConnectCompletedEventArgs Class

xPCTargetPC.ConnectCompleted event data

Syntax

```
public class ConnectCompletedEventArgs : AsyncCompletedEventArgs
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Properties

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

DisconnectCompletedEventArgs Class

xPCTargetPC.DisconnectCompleted event data

Syntax

```
public class DisconnectCompletedEventArgs : AsyncCompletedEventArgs
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Properties

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

GetDataCompletedEventArgs Class

GetDataCompleted event data

Syntax

```
public class GetDataCompletedEventArgs : AsyncCompletedEventArgs
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Properties

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

GetFileScSignalDataObjectCompletedEventArgs Class

xPCDataFileScSignalObject.GetDataCompleted event data

Syntax

```
public class GetFileScSignalDataObjectCompletedEventArgs :
    GetDataCompletedEventArgs
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Properties

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

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

GetHostScSignalDataObjectCompletedEventArgs Class

xPCDataHostScSignalObject.DataObjectCompleted event data

Syntax

```
public class GetHostScSignalDataObjectCompletedEventArgs :
    GetDataCompletedEventArgs
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Properties

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

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

GetLogDataCompletedEventArgs Class

xPCDataLoggingObject.GetDataCompleted event data

Syntax

```
public class GetLogDataCompletedEventArgs :
    GetDataCompletedEventArgs
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Properties

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

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

GetParamCompletedEventArgs Class

xPCParameter.GetParamCompleted event data

Syntax

```
public class GetParamCompletedEventArgs : AsyncCompletedEventArgs
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Properties

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

LoadCompletedEventArgs Class

xPCTargetPC.LoadCompleted event data

Syntax

```
public class LoadCompletedEventArgs : AsyncCompletedEventArgs
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Properties

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

PropertyNotificationEventArgs Class

PropertyNotification event data

Syntax

```
public class PropertyNotificationEventArgs :  
PropertyChangedEventArgs
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Properties

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

RebootCompletedEventArgs Class

xPCTargetPC.RebootCompleted event data

Syntax

```
public class RebootCompletedEventArgs : AsyncCompletedEventArgs
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Properties

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

SetParamCompletedEventArgs Class

xPCParameter.SetParamCompleted event data

Syntax

```
public class SetParamCompletedEventArgs : AsyncCompletedEventArgs
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Properties

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

UnloadCompletedEventArgs Class

xPCTargetPC.UnloadCompleted event data

Syntax

```
public class UnloadCompletedEventArgs : AsyncCompletedEventArgs
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Properties

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

xPCApplication Class

Access to real-time application loaded on target computer

Syntax

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

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Methods

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

Events

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

Properties

Properties	C# Declaration Syntax	Description	Exception
AverageTeT	public double AverageTeT {get;}	Get the average task execution time. The first element contains the average TET number; the second element contains how long it took to achieve the TET time. Task Execution Time (TET) measures how long it takes the kernel to run for one base-rate time step. For a multirate model, use the profiler to find out what the execution time is for each rate.	xPCException — When problem occurs, query xPCException object Reason property.
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 real-time application logging object.	
MaximumTeT	public double MaximumTeT {get;}	Get the maximum task execution time. The first element contains the maximum TET number; the second element contains how	xPCException — When problem occurs, query xPCException object Reason property.

Properties	C# Declaration Syntax	Description	Exception
		long it took to achieve the TET time.	
MinimumTeT	public double MinimumTeT {get;}	Get the minimum task execution time. The first element contains the minimum TET number; the second element contains how long it took to achieve the TET time.	xPCException — When problem occurs, query xPCException object Reason property.
Name	public string Name {get;}	Get the current name of the loaded real-time application	xPCException — When problem occurs, query xPCException object Reason property.
Parameters	public xPCParameters Parameters {get;}	Get reference to the xPCParameters object.	
SampleTime	public double SampleTime {get; set;}	Get or set Sample time Note: Some blocks produce incorrect results when you change their sample time at run time. If you include such blocks in your model, the software displays a warning message during model build. To avoid incorrect results, change the sample time in the original model, and then rebuild and download the model.	xPCException — When problem occurs, query xPCException object Reason property.

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

xPCAppLogger Class

Access to real-time application loggers

Syntax

```
public class xPCAppLogger : xPCApplicationObject
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Properties

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

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

xPCDataFileScSignalObject Class

Object that holds logged file scope signal data

Syntax

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

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

accesses an object that holds logged file scope signal data.

Methods

Method	Description
xPCDataFileScSignalObject.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	<code>public xPCFileScopeSignal ScopeSignalObject {get;}</code>	Get parent scope signal xPCFileScopeSignal object.

xPCDataHostScSignalObject Class

Object that holds logged host scope signal data

Syntax

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

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

accesses an object that holds logged host scope signal data.

Methods

Method	Description
xPCDataHostSc-SignalObject.GetData	Copy host scope signal data from target computer
xPCDataHostSc-SignalObject.-GetDataAsync	Asynchronously copy host scope signal data from target computer

Events

Event	Description
xPCDataHostSc-SignalObject.-GetDataCompleted	Event when xPCDataHostScSignalObject.GetDataAsync is complete

Properties

Property	C# Declaration Syntax	Description
Decimation	<code>public int Decimation {get; set;}</code>	A number n , where every n th sample is acquired in a scope window.
NumSamples	<code>public int NumSamples {get; set;}</code>	<p>Get or set number of contiguous samples captured during the acquisition of a data package. The scope writes data samples into a memory buffer of size <code>NumSamples</code>.</p> <p>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.</p>
ScopeSignal-Object	<code>public xPCHostScopeSignal ScopeSignalObject {get;}</code>	Get parent scope signal <code>xPCHostScopeSignal</code> object.
Startindex	<code>public int StartIndex {get; set;}</code>	Get and set the index of the first sample to retrieve from the log.

xPCDataLoggingObject Class

Object that holds logged data

Syntax

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

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

```
public class xPCDataLoggingObject : xPCApplicationNotificationObject,
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	<code>public int Decimation {get; set;}</code>	A number n , where every n th sample is acquired in a scope window.
LogId	<code>public int LogId {get;}</code>	
NumSamples	<code>public int NumSamples {get; set;}</code>	Get or set number of contiguous samples captured during the acquisition of a data package.
Startindex	<code>public int StartIndex {get; set;}</code>	Get and set the index of the first sample to retrieve from the log.

xPCDirectoryInfo Class

Access folders and subfolders of target computer file system

Syntax

```
public class xPCDirectoryInfo : xPCFileSystemInfo
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

A fully qualified folder name can have a maximum of 248 characters, including the drive letter, colon, and backslash.

Constructor

Constructor	Description
<code>xPCDirectoryInfo</code>	Construct new instance of the <code>xPCDirectoryInfo</code> class on specified path

Methods

Method	Description
<code>xPCDirectoryInfo.-Create</code>	Create folder
<code>xPCDirectoryInfo.-Delete</code>	Delete empty <code>xPCDirectoryInfo</code> object
<code>xPCDirectoryInfo.-GetDirectories</code>	Subfolders of current folder

Method	Description
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 character 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 character string.	xPCException — When problem occurs, query xPCException object Reason property.
Parent	public xPCDirectoryInfo Parent {get;}	Get the parent folder of a specified subfolder.	xPCException — When problem occurs, query xPCException object Reason property.
Root	public xPCDirectoryInfo Root {get;}	Get the root portion of a path.	xPCException — When problem occurs, query xPCException object Reason property.

xPCDriveInfo Class

Information for target computer drive

Syntax

```
public class xPCDriveInfo
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

public class xPCDriveInfo accesses information on a target computer drive.

Constructor

Constructor	Description
xPCDriveInfo	Initialize new instance of xPCDriveInfo class

Methods

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

Properties

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

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

xPCEException Class

Information for xPCEException

Syntax

```
public class xPCEException : Exception, ISerializable
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Constructor

Constructor	Description
xPCEException	Construct new instance of xPCEException class

Properties

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

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

xPCFileInfo Class

Access to file and xPCFileStream objects

Syntax

```
public class xPCDriveInfo
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

There are the following limitations:

- You can have a maximum of eight files open on the target computer at the same time.
- The largest single file that you can create is 4 GB.
- A fully qualified folder name can have a maximum of 248 characters, including the drive letter, colon, and backslash.
- A fully qualified file name can have a maximum of 260 characters. The file part can have 12 characters—a maximum of eight characters for the file name, one character for the period, and a maximum of three characters for the file extension. If the file name is longer than eight characters, the software truncates it to six characters and adds '~1' to the end of the file name.

Constructor

Constructor	Description
xPCFileInfo	Construct new instance of xPCFileInfo class

Methods

Method	Description
<code>xPCFileInfo.CopyToHost</code>	Copy file from target computer file system to development computer file system
<code>xPCFileInfo.Create</code>	Create file in specified path name
<code>xPCFileInfo.Delete</code>	Permanently delete file on target computer
<code>xPCFileInfo.Open</code>	Open file
<code>xPCFileInfo.OpenRead</code>	Create read-only <code>xPCFileStream</code> object
<code>xPCFileInfo.Rename</code>	Rename file

Properties

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

xPCFileScope Class

Access to file scopes

Syntax

```
public class xPCFileScope : xPCScope
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

There are the following limitations:

- You can have a maximum of eight files open on the target computer at the same time.
- The largest single file that you can create is 4 GB.
- A fully qualified folder name can have a maximum of 248 characters, including the drive letter, colon, and backslash.
- A fully qualified file name can have a maximum of 260 characters. The file part can have 12 characters—a maximum of eight characters for the file name, one character for the period, and a maximum of three characters for the file extension. If the file name is longer than eight characters, the software truncates it to six characters and adds '~1' to the end of the file name.

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

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

xPCFileScopeCollection Class

Collection of xPCFileScope objects

Syntax

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

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Methods

Method	Description
xPCFileScopeCollection.Add	Create xPCFileScope object with the next available scope ID as key
xPCFileScopeCollection.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

Access to file scope signals

Syntax

```
public class xPCFileScopeSignal : xPCScopeSignal
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Properties

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

xPCFileScopeSignalCollection Class

Collection of xPCFileScopeSignal objects

Syntax

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

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Methods

Method	Description
xPCFileScope-SignalCollection.Add	Add signals to file scope
xPCFileScope-SignalCollection.-Refresh	Synchronize with signals for associated scope on target computer

Properties

Property	C# Declaration Syntax	Description	Exception
Item	public xPCFileScopeSignal	Get xPCFileScopeSignal object from signal name (<i>blkpath</i>).	xPCException — When problem occurs, query xPCException object Reason property.

Property	C# Declaration Syntax	Description	Exception
	<code>Item[string blkpath] {get;}</code>	<i>blkpath</i> is the signal name that represents a signal object added to its parent xPCHostScope object. This property returns the file scope signal object as type xPCFileScopeSignal.	

xPCFileStream Class

Access xPCFileStream objects

Syntax

```
public class xPCFileStream : IDisposable
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

There are the following limitations:

- You can have a maximum of eight files open on the target computer at the same time.
- The largest single file that you can create is 4 GB.
- A fully qualified folder name can have a maximum of 248 characters, including the drive letter, colon, and backslash.
- A fully qualified file name can have a maximum of 260 characters. The file part can have 12 characters—a maximum of eight characters for the file name, one character for the period, and a maximum of three characters for the file extension. If the file name is longer than eight characters, the software truncates it to six characters and adds '~1' to the end of the file name.

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

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.
```

There are the following limitations:

- You can have a maximum of eight files open on the target computer at the same time.
- The largest single file that you can create is 4 GB.
- A fully qualified folder name can have a maximum of 248 characters, including the drive letter, colon, and backslash.
- A fully qualified file name can have a maximum of 260 characters. The file part can have 12 characters—a maximum of eight characters for the file name, one character for the period, and a maximum of three characters for the file extension. If the file name is longer than eight characters, the software truncates it to six characters and adds '~1' to the end of the file name.

Methods

Method	Description
xPCFileSystem.- CreateDirectory	Create folder
xPCFileSystem.- GetCurrentDirectory	Current working folder for real-time application

Method	Description
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

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.

Property	C# Declaration Syntax	Description
Exists	<code>public abstract bool Exists {get;}</code>	Get value that indicates existence of file or folder.
Extension	<code>public string Extension {get;}</code>	Get character string that represents file extension.
FullName	<code>public virtual string FullName {get;}</code>	Get full path name of file or folder.
Name	<code>public abstract string Name {get;}</code>	Get name of folder.

xPCHostScope Class

Access to host scopes

Syntax

```
public class xPCHostScope : xPCScope
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Methods

The `xPCHostScope` class inherits methods from `xPCScope` Class.

Events

The `xPCHostScope` class inherits events from `xPCScope` Class.

Properties

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

Property	C# Declaration Syntax	Description	Exception
DateTime-Object	<pre>public xPCDataHostSc- SignalObject DateTimeObject {get;}</pre>	Get host scope time data object <code>xPCDataHost-ScSignalObject</code> associated with this scope.	

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

xPCHostScopeCollection Class

Collection of xPCHostScope objects

Syntax

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

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Methods

Method	Description
xPCHostScopeCollection. Add	Create xPCHostScope object with the next available scope ID as key
xPCHostScopeCollection. 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

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	<pre>public xPCDataHostScSignalObject HostScopeSignalDataObject {get;}</pre>	Get host scope signal data object.
Scope	<pre>public xPCHostScope Scope {get;}</pre>	Get host scope.

xPCHostScopeSignalCollection Class

Collection of xPCHostScopeSignal objects

Syntax

```
public class xPCHostScopeSignal : xPCScopeSignal
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Methods

Method	Description
xPCHostScope-SignalCollection.Add	Create xPCHostScopeSignal object
xPCHostScope-SignalCollection.-Refresh	Synchronize signals for associated host scopes on target computer

Properties

Property	C# Declaration Syntax	Description	Exception
Item	public xPCHostScopeSignal	Get xPCHostScopeSignal object from signal name (<i>blkpath</i>).	xPCException — When problem occurs, query xPCException object Reason property.

Property	C# Declaration Syntax	Description	Exception
	<code>Item[string blkpath] {get;}</code>	<p><i>blkpath</i> is the signal name that represents a signal object added to its parent xPCHostScope object.</p> <p>This property returns the file scope signal object as type xPCHostScopeSignal.</p>	

xPCLog Class

Base data logging class

Syntax

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

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Properties

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

xPCOutputLogger Class

Access to output logger

Syntax

```
public class xPCOutputLogger : xPCLog
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Properties

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

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

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

xPCParameter Class

Single run-time tunable parameter

Syntax

```
public class xPCParameter : xPCApplicationNotificationObject
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Methods

Method	Description
<code>xPCParameter.GetParam</code>	Get parameter values from target computer
<code>xPCParameter.-GetParamAsync</code>	Asynchronous request to get parameter values from target computer
<code>xPCParameter.SetParam</code>	Change value of parameter on target computer
<code>xPCParameter.-SetParamAsync</code>	Asynchronous request to change parameter value on target computer

Events

Event	Description
<code>xPCParameter.-GetParamCompleted</code>	Event when <code>xPCParameter.GetParamAsync</code> is complete

Event	Description
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 character string, of the parameter for an instance of an xPCParameter object.	
Dimensions	public int[] Dimensions {get;}	Get an array that contains elements of dimension lengths.	
Name	public string Name {get;}	Get the name of the parameter to an instance of an xPCParameter	
ParameterId	public int ParameterId {get;}	Get the numerical index (identifier) that maps to an instance of an xPCParameter object.	
Rank	public int Rank {get;}	Get the number of dimensions of the parameter	
Value	public Array Value {get; set;}	Get and set the parameter value.	xPCException — When problem occurs, query xPCException object Reason property.

xPCParameters Class

Access run-time parameters

Syntax

```
public class xPCParameters : xPCApplicationObject
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Methods

Method	Description
<code>xPCParameters.-LoadParameterSet</code>	Load parameter values for real-time application
<code>xPCParameters.Refresh</code>	Refresh state of object
<code>xPCParameters.-SaveParameterSet</code>	Save parameter values of real-time application

Properties

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

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

xPCScope Class

Access Simulink Real-Time scopes

Syntax

```
public abstract class xPCScope : xPCApplicationNotificationObject
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Methods

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

Events

Event	Description
<code>xPCScope.ScopeStarted</code>	Event after <code>xPCScope.Start</code> is complete
<code>xPCScope.ScopeStarting</code>	Event before <code>xPCScope.Start</code> executes
<code>xPCScope.ScopeStopped</code>	Event after <code>xPCScope.Stop</code> is complete
<code>xPCScope.ScopeStopping</code>	Event before <code>xPCScope.Stop</code> executes

Properties

Property	C# Declaration Syntax	Description	Exception
Decimation	public int Decimation {get; set;}	Get or set a number n , where every n th sample is acquired in a scope window.	xPCException — When problem occurs, query xPCException object Reason property.
NumPrePost-Samples	public int NumPrePostSamples {get; set;}	Get or set number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set TriggerMode to 'FreeRun', changing this property does not change data acquisition.	xPCException — When problem occurs, query xPCException object Reason property.
NumSamples	public int NumSamples {get; set;}	Get or set number of contiguous samples captured during the acquisition of a data package. The scope writes data samples into a memory buffer of size NumSamples. 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	xPCException — When problem occurs, query xPCException object Reason property.

Property	C# Declaration Syntax	Description	Exception
		is possible that your data contains zeroes.	
ScopeId	public int ScopeId {get;}	A numeric index, unique for each scope.	
Status	public SCSTATUS Status {get;}	Indicate whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are 'Acquiring', 'Ready for being Triggered', 'Interrupted', and 'Finished'.	xPCException — When problem occurs, query xPCException object Reason property.
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	<pre>public SCTRIGGERMODE TriggerMode {get; set;}</pre>	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	<pre>public int TriggerScope {get; set;}</pre>	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	<pre>public int TriggerScopeSample {get; set;}</pre>	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.
TriggerSlope	<pre>public TRIGGERSLOPE {get; set;}</pre>	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.

Property	C# Declaration Syntax	Description	Exception
Type	<code>public string Type {get;}</code>	Get scope type as a character string.	

For file scopes, the `NumSamples` parameter works with the `autorestart` parameter.

- **Autorestart is on** — When the scope triggers, the scope collects data into a memory buffer. A background task examines the buffer and writes data to the disk continuously, appending new data to the end of the file. When the scope reaches the number of samples that you specified, it starts collecting data again, overwriting the memory buffer. If the background task cannot keep pace with data collection, data can be lost.
- **Autorestart is off** — When the scope triggers, the scope collects data into a memory buffer up to the number of samples that you specified, and then stops. A background task examines the buffer and writes data to the disk continuously, appending the new data to the end of the file.

xPCScopeCollectionEventArgs Class

xPCScopeCollection.Added event data

Syntax

```
public class xPCScopeCollectionEventArgs : EventArgs
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Properties

Properties	C# Declaration Syntax	Description
Scope	<pre>public xPCScope Scope {get;}</pre>	Get xPCScope object you added.

xPCScopeRemCollectionEventArgs Class

xPCScopeCollection.Removed event data

Syntax

```
public class xPCScopeRemCollectionEventArgs : EventArgs
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Properties

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

xPCScopeSignalCollectionEventArgs Class

xPCScopeSignalCollection.Added event data

Syntax

```
public class xPCScopeSignalCollectionEventArgs : EventArgs
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Properties

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

xPCScopes Class

Access scope objects

Syntax

```
public class xPCScopes : xPCApplicationObject
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Methods

Method	Description
<code>xPCScopes.RefreshAll</code>	Synchronize with all scopes on target computer

Properties

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

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

xPCSignal Class

Access signal objects

Syntax

```
public class xPCSignal : xPCApplicationObject
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Methods

Method	Description
<code>xPCSignal.GetValue</code>	Value of signal at moment of request
<code>xPCSignal.TryGetValue</code>	Status of get signal value at moment of request

Properties

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

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

xPCSignals Class

Access signal objects

Syntax

```
public class xPCSignals : xPCApplicationObject
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Methods

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

Properties

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

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

xPCStateLogger Class

Access to state log

Syntax

```
public class xPCStateLogger : xPCLog
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Properties

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

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

xPCTargetPC Class

Access target computer

Syntax

```
public xPCTargetPC()
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Note: RS-232 communication type has been removed. Configure TCP/IP communication instead.

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

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

Events

Event	Description
xPCTargetPC.- 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

Event	Description
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 real-time application. If no communication is established, the property returns a NULL object.	
Communication-TimeOut	public int CommunicationTimeout {get; set;}	Get or set the communication timeout in seconds.	xPCException — When problem occurs, query xPCException object Reason property.
Component	public IComponent Component {get;}	Get component associated with the ISite when implemented by a class.	
Container	public IContainer Container {get;}	Get the IContainer associated with the ISite when implemented by a class.	
Container-Control	public ContainerControl	Provide focus-management functionality for controls	

Property	C# Declaration Syntax	Description	Exception
	<code>ContainerControl {get; set;}</code>	that can function as containers for other controls.	
DLMFileName	<code>public string DLMFileName {get; set;}</code>	Get or set the full path to the DLM file name.	
Echo	<code>public bool Echo {get; set;}</code>	Get or set the target display on the target computer.	xPCException — When problem occurs, query xPCException object Reason property.
FileSystem	<code>public xPCFileSystem FileSystem {get;}</code>	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	<code>public XPCProtocol HostTargetComm {get; set;}</code>	Get or set the physical medium for communication. See xPCProtocol Enumerated Data Type . Setting HostTarget-Comm to RS232 has no effect. Value remains set to TCPIP .	
IsConnected	<code>public bool IsConnected {get;}</code>	Get connection status (established or not) to a remote target computer.	
IsConnecting-Busy	<code>public bool IsConnectingBusy {get;}</code>	Get ConnectAsync request status (in progress or not).	

Property	C# Declaration Syntax	Description	Exception
IsDisconnectingBusy	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).	
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.	

Property	C# Declaration Syntax	Description	Exception
TcpIpTarget-Port	<pre>public string TcpIpTargetPort {get; set;}</pre>	Get or set the TCP/IP target port. The default is 22222 and should not cause problems. This number is higher than the reserved area (for example, the port numbers reserved for telnet or ftp). The software uses this value only for the target computer.	

xPCTargetScope Class

Access to target scopes

Syntax

```
public class xPCTargetScope : xPCScope
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Methods

The `xPCTargetScope` class inherits methods from `xPCScope` Class.

Events

The `xPCTargetScope` class inherits events from `xPCScope` Class.

Properties

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

Property	C# Declaration Syntax	Description	Exception
Display-Mode	<pre>public SCDISPLAYMODE DisplayMode {get; set;}</pre>	Get or set scope mode for displaying signals.	<code>xPCException</code> — When problem occurs, query <code>xPCException</code> object <code>Reason</code> property.

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

xPCTargetScopeCollection Class

Collection of xPCTargetScope objects

Syntax

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

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Methods

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

xPCTargetScopeSignalCollection Class

Collection of xPCHostScopeSignal objects

Syntax

```
public class xPCTargetScopeSignalCollection :
    xPCScopeSignalCollection
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Methods

Method	Description
xPCTargetScope-SignalCollection.Add	Create xPCTargetScopeSignal object
xPCTargetScope-SignalCollection.-Refresh	Synchronize signals for associated target scopes on target computer

Properties

Property	C# Declaration Syntax	Description	Exception
Item	public xPCTgtScopeSignal	Get xPCTgtScopeSignal object from signal name (<i>blkpath</i>).	xPCException — When problem occurs, query

Property	C# Declaration Syntax	Description	Exception
	<pre>Item[string blkpath] {get;}</pre>	<p><i>blkpath</i> is the signal name that represents a signal object added to its parent xPCTargetScope object.</p> <p>This property returns the file scope signal object as type xPCTgtScopeSignal.</p>	xPCException object Reason property.

xPCTETLogger Class

Access to task execution time (TET) logger

Syntax

```
public class xPCTETLogger : xPCLog
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Properties

The xPCTETLogger class inherits its other properties from xPCLog Class.

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

xPCTgtScopeSignal Class

Access to target scope signals

Syntax

```
public class xPCTgtScopeSignal : xPCScopeSignal
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Properties

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

xPCTimeLogger Class

Access to output log

Syntax

```
public class xPCTimeLogger : xPCLog
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Properties

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

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

xPCFileInfo.Open

Open file

Syntax

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

Description

Class: xPCFileInfo Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

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

xPCFileInfo.OpenRead

Create read-only xPCFileStream object

Syntax

```
public xPCFileStream OpenRead()
```

Description

Class: xPCFileInfo Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

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

xPCTargetPC.Ping

Test communication between development and target computers

Syntax

```
public bool Ping()
```

Description

Class: xPCTargetPC Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCFileStream.Read

Read block of bytes from stream and write data to buffer

Syntax

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

Description

Class: xPCFileStream Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

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

xPCTargetPC.Reboot

Restart target computer

Syntax

```
public void Reboot()
```

Description

Class: xPCTargetPC Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

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

xPCTargetPC.RebootAsync

Asynchronous request to restart target computer

Syntax

```
public void RebootAsync()
```

Description

Class: xPCTargetPC Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

Exception	Condition
InvalidOperationException	When another thread uses this method.

xPCTargetPC.RebootCompleted

Event when xPCTargetPC.RebootAsync is complete

Syntax

```
public event RebootCompletedEventHandler RebootCompleted
```

Description

Class: xPCTargetPC Class

Event

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCTargetPC.Rebooted

Event after xPCTargetPC.Reboot is complete

Syntax

```
public event EventHandler Rebooted
```

Description

Class: xPCTargetPC Class

Event

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCTargetPC.Rebooting

Event before xPCTargetPC.Reboot starts

Syntax

```
public event EventHandler Rebooting
```

Description

Class: xPCTargetPC Class

Event

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

public event EventHandler Rebooting occurs before a restart operation executes.

xPCFileScopeCollection.Refresh

Synchronize with file scopes on target computer

Syntax

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

Description

Class: xPCFileScopeCollection Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

```
Overrides xPCScopeCollection<xPCFileScope>.Refresh().
```

xPCScopes.RefreshAll

Refresh state of object

Syntax

```
public void RefreshAll()
```

Description

Class: xPCScopes Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCDriveInfo.Refresh

Synchronize with file drives on target computer

Syntax

```
public void Refresh()
```

Description

Class: xPCDriveInfo Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCFileScopeSignalCollection.Refresh

Synchronize with signals for associated scope on target computer

Syntax

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

Description

Class: xPCFileScopeSignalCollection Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Overrides `xPCScopeCollection<xPCFileScopeSignal>.Refresh()`.

Exception

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

xPCHostScopeCollection.Refresh

Refresh host scope object state

Syntax

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

Description

Class: xPCHostScopeCollection Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Overrides xPCScopeCollection<xPCHostScope>.Refresh().

Exception

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

xPCHostScopeSignalCollection.Refresh

Synchronize signals for associated host scopes on target computer

Syntax

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

Description

Class: xPCHostScopeSignalCollection Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Overrides xPCScopeCollection<xPCHostScope>.Refresh().

Exception

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

xPCParameters.Refresh

Refresh state of object

Syntax

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

Description

Class: xPCParameters Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCSignals.Refresh

Refresh state of object

Syntax

```
public void Refresh()
```

Description

Class: xPCSignals Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCTargetScopeCollection.Refresh

Refresh target scope object state

Syntax

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

Description

Class: xPCTargetScopeCollection Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Overrides xPCScopeCollection<xPCTargetScope>.Refresh().

xPCTargetScopeSignalCollection.Refresh

Synchronize signals for associated target scopes on target computer

Syntax

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

Description

Class: xPCTargetScopeSignalCollection Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Overrides `xPCScopeSignalCollection<xPCTgtScopeSignal>.Refresh()`.

Exception

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

xPCFileSystem.RemoveFile

Remove file name from target computer

Syntax

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

Description

Class: xPCFileSystem Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

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

xPCFileInfo.Rename

Rename file

Syntax

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

Description

Class: xPCFileInfo Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

A fully qualified file name can have a maximum of 260 characters. The file part can have 12 characters—a maximum of eight characters for the file name, one character for the period, and a maximum of three characters for the file extension. If the file name is longer than eight characters, the software truncates it to six characters and adds '~1' to the end of the file name.

Exception

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

xPCParameters.SaveParameterSet

Save parameter values of real-time application

Syntax

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

Description

Class: xPCParameters Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

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

SCDISPLAYMODE Enumerated Data Type

Target scope display mode values

Syntax

```
public enum SCDISPLAYMODE
```

Description

Enumerated Data Type

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Members

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

SCFILEMODE Enumerated Data Type

Write mode values for when file allocation table entry is updated

Syntax

```
public enum SCFILEMODE
```

Description

Enumerated Data Type

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Members

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

xPCScope.ScopeStarted

Event after xPCScope.Start is complete

Syntax

```
public event EventHandler ScopeStarted
```

Description

Class: xPCScope Class

Event

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCScope.ScopeStarting

Event before xPCScope.Start executes

Syntax

```
public event EventHandler ScopeStarting
```

Description

Class: xPCScope Class

Event

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

public event EventHandler ScopeStarting occurs before a scope executes.

xPCScope.ScopeStopped

Event after xPCScope.Stop is complete

Syntax

```
public event EventHandler ScopeStarting
```

Description

Class: xPCScope Class

Event

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCScope.ScopeStopping

Event before xPCScope . Stop executes

Syntax

```
public event EventHandler ScopeStopping
```

Description

Class: xPCScope Class

Event

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

SCSTATUS Enumerated Data Type

Scope status values

Syntax

```
public enum SCSTATUS
```

Description

Enumerated Data Type

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

`public enum SCSTATUS` specifies scope status values.

Members

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

SCTRIGGERMODE Enumerated Data Type

Scope trigger mode values

Syntax

```
public enum SCTRIGGERMODE
```

Description

Enumerated Data Type

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

public enum SCTRIGGERMODE specifies scope trigger mode values.

Members

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

SCTRIGGERSLOPE Enumerated Data Type

Scope trigger slope values

Syntax

```
public enum SCTRIGGERSLOPE
```

Description

Enumerated Data Type

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

public enum SCTRIGGERSLOPE specifies scope trigger slope values.

Members

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

SCTYPE Enumerated Data Type

Scope type

Syntax

```
public enum SCTYPE
```

Description

Enumerated Data Type

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

`public enum SCTYPE` specifies scope type.

Members

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

xPCFileSystem.SetCurrentDirectory

Current folder

Syntax

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

Description

Class: xPCFileSystem Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

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

xPCParameter.SetParam

Change value of parameter on target computer

Syntax

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

Description

Class: xPCParameter Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

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

xPCParameter.SetParamAsync

Asynchronous request to change parameter value on target computer

Syntax

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

Description

Class: xPCParameter Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

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

Exception

Exception	Condition
InvalidOperationException	When another thread uses this method.

xPCParameter.SetParamCompleted

Event when `xPCParameter.SetParamAsync` is complete

Description

Class: `xPCParameter` Class

Event

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: `C#`

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

xPCApplication.Start

Start real-time application execution

Syntax

```
public void Start()
```

Description

Class: xPCApplication Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

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

xPCFileScopeCollection.StartAll

Start all file scopes in one call

Syntax

```
public void StartAll()
```

Description

Class: xPCFileScopeCollection Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCHostScopeCollection.StartAll

Start all host scopes in one call

Syntax

```
public void StartAll()
```

Description

Class: xPCHostScopeCollection Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

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

xPCTargetScopeCollection.StartAll

Start all target scopes in one call

Syntax

```
public void StartAll()
```

Description

Class: xPCTargetScopeCollection Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCScope.Start

Start scope

Syntax

```
public void Start()
```

Description

Class: xPCScope Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

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

xPCApplication.Started

Event after xPCApplication.Start is complete

Syntax

```
public event EventHandler Started
```

Description

Class: xPCApplication Class

Event

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCApplication.Starting

Event before xPCApplication.Start executes

Syntax

```
public event EventHandler Starting
```

Description

Class: xPCApplication Class

Event

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCApplication.Stop

Stop real-time application execution

Syntax

```
public void Stop()
```

Description

Class: xPCApplication Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

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

xPCFileScopeCollection.StopAll

Stop all file scopes in one call

Syntax

```
public void StopAll()
```

Description

Class: xPCFileScopeCollection Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCHostScopeCollection.StopAll

Stop all host scopes in one call

Syntax

```
public void StopAll()
```

Description

Class: xPCHostScopeCollection Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

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

xPCTargetScopeCollection.StopAll

Stop all target scopes in one call

Syntax

```
public void StopAll()
```

Description

Class: xPCTargetScopeCollection Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCScope.Stop

Stop scope

Syntax

```
public void Stop()
```

Description

Class: xPCScope Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

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

xPCApplication.Stopped

Event after xPCApplication.Stop is complete

Syntax

```
public event EventHandler Stopped
```

Description

Class: xPCApplication Class

Event

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCApplication.Stopping

Event before xPCApplication.Stop executes

Syntax

```
public event EventHandler Stopping
```

Description

Class: xPCApplication Class

Event

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCTargetPC.tcpPing

Determine TCP/IP accessibility of remote computer

Syntax

```
public bool tcpPing()
```

Description

Class: xPCTargetPC Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCScope.Trigger

Software-trigger start of data acquisition for scope

Syntax

```
public void Trigger()
```

Description

Class: xPCScope Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

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

xPCSignal.TryGetValue

Status of get signal value at moment of request

Syntax

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

Description

Class: xPCSignal Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCTargetPC.Unload

Unload real-time application from target computer

Syntax

```
public void Unload()
```

Description

Class: xPCTargetPC Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

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

xPCTargetPC.UnloadAsync

Asynchronous request to unload real-time application from target computer

Syntax

```
public void UnloadAsync()
```

Description

Class: xPCTargetPC Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

Exception	Condition
InvalidOperationException	When another thread uses this method.

xPCTargetPC.UnloadCompleted

Event when xPCTargetPC.UnloadAsync is complete

Syntax

```
public event UnloadCompletedEventHandler UnloadCompleted
```

Description

Class: xPCTargetPC Class

Event

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCTargetPC.Unloaded

Event after xPCTargetPC.Unload is complete

Syntax

```
public event EventHandler Unloaded
```

Description

Class: xPCTargetPC Class

Event

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCTargetPC.Unloading

Event before xPCTargetPC.Unload starts

Syntax

```
public event EventHandler Unloading
```

Description

Class: xPCTargetPC Class

Event

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCFileStream.Write

Write block of bytes to file stream

Syntax

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

Description

Class: xPCFileStream Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

Exception	Condition
xPCException	When problem occurs, query xPCException object <code>Reason</code> property.

xPCFileStream.WriteByte

Write byte to current position in file stream

Syntax

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

Description

Class: xPCFileStream Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

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

xPCAppStatus Enumerated Data Type

Real-time application status return values

Syntax

```
public enum xPCAppStatus
```

Description

Enumerated Data Type

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Members

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

xPCDirectoryInfo

Construct new instance of xPCDirectoryInfo class on specified path

Syntax

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

Description

Class: xPCDirectoryInfo Class

Constructor

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

A fully qualified folder name can have a maximum of 248 characters, including the drive letter, colon, and backslash.

Exception

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

xPCDriveInfo

Construct new instance of xPCDriveInfo class

Syntax

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

Description

Class: xPCDriveInfo Class

Constructor

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Exception

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

xPCEException

Construct new instance of xPCEException class

Syntax

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

Description

Class: xPCEException Class

Constructor

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

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

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

`public xPCEException(SerializationInfo info, StreamingContext context)` initializes a new instance of the xPCEException 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 character string. *tgt* is the `xPCTargetPC` object that raised the error.

xPCExceptionReason Enumerated Data Type

Exception reasons

Syntax

```
public enum xPCExceptionReason
```

Description

Enumerated Data Type

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

`public enum xPCExceptionReason` specifies the reasons for an exception. See “C API Error Messages” on page 1-8 for definitions.

xPCFileInfo

Construct new instance of `xPCFileInfo` class

Syntax

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

Description

Class: `xPCFileInfo` Class

Constructor

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public xPCFileInfo(xPCTargetPC tgt, string fileName)` initializes a new instance of the `xPCFileInfo` class. *tgt* is an `xPCTargetPC` object that represents the target computer for which you want to return the file information. *fileName* is a character 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.

There are the following limitations:

- You can have a maximum of eight files open on the target computer at the same time.
- The largest single file that you can create is 4 GB.
- A fully qualified folder name can have a maximum of 248 characters, including the drive letter, colon, and backslash.
- A fully qualified file name can have a maximum of 260 characters. The file part can have 12 characters—a maximum of eight characters for the file name, one character for the period, and a maximum of three characters for the file extension. If the file name is longer than eight characters, the software truncates it to six characters and adds '~1' to the end of the file name.

Exception

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

xPCFileMode Enumerated Data Type

Open file with permissions

Syntax

```
public enum xPCFileMode
```

Description

Enumerated Data Type

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Members

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

xPCFileStream

Construct new instance of xPCFileStream class

Syntax

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

Description

Class: xPCFileStream Class

Method

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

There are the following limitations:

- You can have a maximum of eight files open on the target computer at the same time.
- The largest single file that you can create is 4 GB.
- A fully qualified folder name can have a maximum of 248 characters, including the drive letter, colon, and backslash.
- A fully qualified file name can have a maximum of 260 characters. The file part can have 12 characters—a maximum of eight characters for the file name, one character for the period, and a maximum of three characters for the file extension. If the file name is longer than eight characters, the software truncates it to six characters and adds '~1' to the end of the file name.

Exception

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

xPCFileSystemInfo

Construct new instance of xPCFileSystemInfo class

Syntax

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

Description

Class: xPCFileSystemInfo Class

Constructor

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

xPCLogMode Enumerated Data Type

Specify log mode values

Syntax

```
public enum xPCLogMode
```

Description

Enumerated Data Type

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

`public enum xPCLogMode` specifies log mode values.

Members

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

xPCLogType Enumerated Data Type

Logging type values

Syntax

```
public enum xPCLogType
```

Description

Namespace: MathWorks.xPCTarget.FrameWork

Enumerated Data Type

Syntax Language: C#

`public enum xPCLogType` specifies logging type values.

Members

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

xPCProtocol Enumerated Data Type

Development computer and target computer communication medium

Syntax

```
public enum XPCProtocol
```

Description

Enumerated Data Type

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Note: RS-232 communication type has been removed. Configure TCP/IP communication instead.

Members

Member	Description
TCPIP	Ethernet link

xPCTargetPC

Construct new instance of xPCTargetPC class

Syntax

```
public xPCTargetPC()
```

Description

Class: xPCTargetPC Class

Constructor

Namespace: MathWorks.xPCTarget.FrameWork

Syntax Language: C#

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

Simulink Real-Time API for C

Using the C API

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

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

```
close(slrt)
```

This frees the connection to the target computer for use by your Simulink Real-Time C API application. Conversely, you will need to quit your application, or do the equivalent of calling the function `xPCClosePort`, to access the target from a MATLAB session.

- The Simulink Real-Time C API functions that communicate with the target computer check for timeouts during communication. If the TCP/IP connection times out, these functions will exit with the global variable `xPCError` set to `ETCPTIMEOUT`. Use the `xPCGetLoadTimeout` and `xPCSetLoadTimeout` functions to get and set the timeout values, respectively.

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

- Almost every function (except `xPCOpenTcpIpPort`, `xPCGetLastError`, and `xPCErrorMsg`) has as one of its parameters the integer variable *port*. This variable is returned by `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 character string about the error by using the function `xPCErrorMsg`. Although the actual error values are subject to change, a zero value typically means that the operation completed without producing an error, while a nonzero value typically signifies an error condition. Note also that the library resets the error value every time an API function is called; therefore, your application should check the error status as soon as possible after a function call.

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

Simulink Real-Time API Reference for C

dirStruct

Type definition for file system folder information structure

Syntax

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

Fields

<i>Name</i>	This value contains the name of the file or folder. A fully qualified folder name can have a maximum of 248 characters, including the drive letter, colon, and backslash.
<i>Ext</i>	This value contains the file type of the element, if the element is a file (<i>isDir</i> is 0). If the element is a folder (<i>isDir</i> is 1), this field is empty.
<i>Day</i>	This value contains the day the file or folder was last modified.
<i>Month</i>	This value contains the month the file or folder was last modified.
<i>Year</i>	This value contains the year the file or folder was last modified.
<i>Hour</i>	This value contains the hour the file or folder was last modified.

<i>Min</i>	This value contains the minute the file or folder was last modified.
<i>isDir</i>	This value indicates if the element is a file (0) or folder (1). If it is a folder, Bytes has a value of 0.
<i>Size</i>	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

`xPCFSDirItems`

diskinfo

Type definition for file system disk information structure

Syntax

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

Fields

<i>Label</i>	This value contains the zero-terminated character string that contains the volume label. The character string is empty if the volume has no label.
<i>DriveLetter</i>	This value contains the drive letter, in uppercase.
<i>Reserved</i>	Reserved.
<i>SerialNumber</i>	This value contains the volume serial number.
<i>FirstPhysicalSector</i>	This value contains the logical block addressing (LBA) address of the logical drive boot record. For 3.5-inch disks, this value is 0.

<i>FATType</i>	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.
<i>FATCount</i>	This value contains the number of FAT partitions on the volume.
<i>MaxDirEntries</i>	This value contains the size of the root folder. For FAT-32 systems, this value is 0.
<i>BytesPerSector</i>	This value contains the sector size. This value is most likely to be 512.
<i>SectorsPerCluster</i>	This value contains, in sectors, the size of the smallest unit of storage that can be allocated to a file.
<i>TotalClusters</i>	This value contains the number of file storage clusters on the volume.
<i>BadClusters</i>	This value contains the number of clusters that have been marked as bad. These clusters are unavailable for file storage.
<i>FreeClusters</i>	This value contains the number of clusters that are currently available for storage.
<i>Files</i>	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.
<i>FileChains</i>	This value contains the number of contiguous cluster chains. On a completely defragmented volume, this value is identical to the value of Files .
<i>FreeChains</i>	This value contains the number of contiguous cluster chains of free clusters. On a completely defragmented volume, this value is 1.
<i>LargestFreeChain</i>	This value contains the maximum allocated file size, in number of clusters, for a newly allocated contiguous file. On a completely defragmented volume, this value is identical to FreeClusters .

Description

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

See Also

xPCFSDiskInfo

fileinfo

Type definition for file information structure

Syntax

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

Fields

<i>FilePos</i>	This value contains the current file pointer.
<i>AllocatedSize</i>	This value contains the currently allocated file size.
<i>ClusterChains</i>	This value indicates how many separate cluster chains are allocated for the file.
<i>VolumeSerialNumber</i>	This value holds the serial number of the volume the file resides on.
<i>FullName</i>	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.

There are the following limitations:

- You can have a maximum of eight files open on the target computer at the same time.
- The largest single file that you can create is 4 GB.
- A fully qualified folder name can have a maximum of 248 characters, including the drive letter, colon, and backslash.

- A fully qualified file name can have a maximum of 260 characters. The file part can have 12 characters—a maximum of eight characters for the file name, one character for the period, and a maximum of three characters for the file extension. If the file name is longer than eight characters, the software truncates it to six characters and adds '~1' to the end of the file name.

See Also

`xPCFSFileInfo`

lgmode

Type definition for logging options structure

Syntax

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

Fields

mode

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

incrementvalue

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

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

Description

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

See Also

`xPCSetLogMode` | `xPCGetLogMode`

scopedata

Type definition for scope data structure

Syntax

```
typedef struct {  
    int    number;  
    int    type;  
    int    state;  
    int    signals[20];  
    int    numsamples;  
    int    decimation;  
    int    triggermode;  
    int    numprepostsamples;  
    int    triggersignal  
    int    triggerscope;  
    int    triggerscopesample;  
    double triggerlevel;  
    int    triggerslope;  
} scopedata;
```

Fields

<i>number</i>	The scope number.										
<i>type</i>	Determines whether the scope is displayed on the development computer or on the target computer. Values are one of the following: <table><tr><td>1</td><td>Host</td></tr><tr><td>2</td><td>Target</td></tr></table>	1	Host	2	Target						
1	Host										
2	Target										
<i>state</i>	Indicates the scope state. Values are one of the following: <table><tr><td>0</td><td>Waiting to start</td></tr><tr><td>1</td><td>Scope is waiting for a trigger</td></tr><tr><td>2</td><td>Data is being acquired</td></tr><tr><td>3</td><td>Acquisition is finished</td></tr><tr><td>4</td><td>Scope is stopped (interrupted)</td></tr></table>	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)
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
<i>signals</i>		List of signal indices from the target object to display on the scope.
		Target scopes are restricted to 10 signals.
<i>numsamples</i>		Number of contiguous samples captured during the acquisition of a data package.
<i>decimation</i>		A number, N, meaning every Nth sample is acquired in a scope window.
<i>triggermode</i>		Trigger mode for a scope. Values are one of the following:
	0	FreeRun (default)
	1	Software
	2	Signal
	3	Scope
<i>numprepostsamples</i>		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.
<i>triggersignal</i>		If <i>triggermode</i> is 2 (Signal), identifies the block output signal to use for triggering the scope. Identify the signal with a signal index.
<i>triggerscope</i>		If <i>triggermode</i> is 3 (Scope), identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered.
<i>triggerscopesample</i>		If <i>triggermode</i> 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.
<i>triggerlevel</i>		If <i>triggermode</i> 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.
<i>triggerslope</i>		If <i>triggermode</i> 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

`xPCSetScope` | `xPCGetScope` | `xPCScGetType` | `xPCScGetState` |
`xPCScGetSignals` | `xPCScGetNumSamples` | `xPCScGetDecimation`
| `xPCScGetTriggerMode` | `xPCScGetNumPrePostSamples` |
`xPCScGetTriggerSignal` | `xPCScGetTriggerScope` | `xPCScGetTriggerLevel` |
`xPCScGetTriggerSlope`

xPCAddScope

Create new scope

Prototype

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

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>scType</i>	Enter the type of scope.
<i>scNum</i>	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

`xPCScAddSignal` | `xPCScRemSignal` | `xPCRemScope` | `xPCSetScope` | `xPCGetScope`
| `xPCGetScopes` | Real-Time Application | Real-Time Application Properties

xPCAverageTET

Return average task execution time

Prototype

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

Arguments

port Enter the value returned by the function xPCOpenTcpIpPort.

Return

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

Description

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

Task Execution Time (TET) measures how long it takes the kernel to run for one base-rate time step. For a multirate model, use the profiler to find out what the execution time is for each rate.

See Also

xPCMaximumTET | xPCMinimumTET | Real-Time Application | Real-Time Application Properties

xPCCloseConnection

Close TCP/IP communication connection

Prototype

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

Arguments

port Enter the value returned by the function xPCOpenTcpIpPort.

Description

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

Note: RS-232 communication type has been removed. Configure TCP/IP communication instead.

See Also

xPCOpenConnection | xPCOpenTcpIpPort | xPCReOpenPort |
xPCRegisterTarget | xPCDeRegisterTarget

xPCClosePort

Close TCP/IP communication connection

Prototype

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

Arguments

port Enter the value returned by the function xPCOpenTcpIpPort.

Description

The xPCClosePort function closes the TCP/IP communication channel opened by xPCOpenTcpIpPort. Calling this function is equivalent to calling xPCCloseConnection and xPCDeRegisterTarget.

Note: RS-232 communication type has been removed. Configure TCP/IP communication instead.

See Also

xPCOpenTcpIpPort | xPCReOpenPort | xPCOpenConnection | xPCCloseConnection | xPCRegisterTarget | xPCDeRegisterTarget | Real-Time Application | Real-Time Application Properties

xPCDeRegisterTarget

Delete target communication properties from Simulink Real-Time API library

Prototype

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

Arguments

port Enter the value returned by 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

`xPCRegisterTarget` | `xPCOpenTcpIpPort` | `xPCClosePort` | `xPCReOpenPort` | `xPCOpenConnection` | `xPCCloseConnection` | `xPCTargetPing`

xPCErrorMsg

Return text description for error message

Prototype

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

Arguments

error_number Enter the constant of an error.

error_message The xPCErrorMsg function copies the error message character 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 character string.

Return

The xPCErrorMsg function returns a character 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

`xPCSetLastError` | `xPCGetLastError`

xPCFreeAPI

Unload Simulink Real-Time DLL

Prototype

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

Description

The `xPCFreeAPI` function unloads the Simulink Real-Time dynamic link library. You must execute this function once at the end of your custom program 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 program.

See Also

`xPCInitAPI` | `xPCNumLogWraps` | `xPCNumLogSamples` | `xPCMaxLogSamples` | `xPCGetStateLog` | `xPCGetTETLog` | `xPCSetLogMode` | `xPCGetLogMode`

xPCFSCD

Change current folder on target computer to specified path

Prototype

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

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>dir</i>	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

`xPCFSGetPWD` | File System

xPCFSCloseFile

Close file on target computer

Prototype

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

Arguments

<i>port</i>	Enter the value returned by the function xPCOpenTcpIpPort.
<i>fileHandle</i>	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

xPCFSOpenFile | xPCFSReadFile | xPCFSWriteFile | File System

xPCFSDir

Get contents of specified folder on target computer

Prototype

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

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>path</i>	Enter the path on the target computer.
<i>data</i>	The contents of the folder are stored in <i>data</i> , whose allocated size is specified in <i>numbytes</i> .
<i>numbytes</i>	Enter the size, in bytes, of the array <i>data</i> .

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

`xPCFSDirSize` | File System

xPCFSDirItems

Get contents of specified folder on target computer

Prototype

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

Arguments

<i>port</i>	Enter the value returned by the function xPCOpenTcpIpPort.
<i>path</i>	Enter the path on the target computer.
<i>dirs</i>	Enter the structure to contain the contents of the folder.
<i>numDirItems</i>	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

dirStruct | File System | xPCFSDirStructSize

xPCFSDirSize

Return size of specified folder listing on target computer

Prototype

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

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>path</i>	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

File System | `xPCFSDirItems`

xPCFSDirStructSize

Get number of items in folder

Prototype

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

Arguments

port Enter the value returned by 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

xPCFSDir | File System

xPCFSDiskInfo

Information about target computer file system

Prototype

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

Arguments

port

Enter the value returned by the function `xPCOpenTcpIpPort`.

driveletter

Enter the drive letter of the file system for which you want information, for example 'C:\'.

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

File System

xPCFSFileInfo

Return information for open file on target computer

Prototype

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

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>fileHandle</i>	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

File System

xPCFSGetError

Get text description for error number on target computer file system

Prototype

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

Arguments

<i>port</i>	Enter the value returned by the function xPCOpenTcpIpPort.
<i>error_number</i>	Enter the constant of an error.
<i>error_message</i>	The character string of the message associated with the error <i>error_number</i> is stored in <i>error_message</i> .

Description

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

xPCFSGetFileSize

Return size of file on target computer

Prototype

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

Arguments

<i>port</i>	Enter the value returned by the function xPCOpenTcpIpPort.
<i>fileHandle</i>	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

xPCFSOpenFile | xPCFSReadFile | File System

xPCFSGetPWD

Get current folder of target computer

Prototype

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

Arguments

<i>port</i>	Enter the value returned by the function xPCOpenTcpIpPort.
<i>pwd</i>	The path of the current folder is stored in <i>pwd</i> .

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 System

xPCFSMKDIR

Create new folder on target computer

Prototype

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

Arguments

port Enter the value returned by the function `xPCOpenTcpIpPort`.

dirname Enter the name of the folder to create on the target computer.

A fully qualified folder name can have a maximum of 248 characters, including the drive letter, colon, and backslash.

Description

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

See Also

`xPCFSGetPWD` | File System

xPCFSOpenFile

Open file on target computer

Prototype

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

Arguments

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

There are the following limitations:

- You can have a maximum of eight files open on the target computer at the same time.
- The largest single file that you can create is 4 GB.
- A fully qualified folder name can have a maximum of 248 characters, including the drive letter, colon, and backslash.

- A fully qualified file name can have a maximum of 260 characters. The file part can have 12 characters—a maximum of eight characters for the file name, one character for the period, and a maximum of three characters for the file extension. If the file name is longer than eight characters, the software truncates it to six characters and adds '~1' to the end of the file name.

See Also

xPCFSCloseFile | xPCFSGetFileSize | xPCFSReadFile | xPCFSWriteFile | File System

xPCFSReadFile

Read open file on target computer

Prototype

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

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>fileHandle</i>	Enter the file handle of an open file on the target computer.
<i>start</i>	Enter an offset from the beginning of the file from which this function can start to read.
<i>numbytes</i>	Enter the number of bytes this function is to read from the file.
<i>data</i>	The contents of the file are stored in <i>data</i> .

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

`xPCFSCloseFile` | `xPCFSGetFileSize` | `xPCFSOpenFile` | `xPCFSWriteFile` | File System

xPCFSRemoveFile

Remove file from target computer

Prototype

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

Arguments

<i>port</i>	Enter the value returned by the function xPCOpenTcpIpPort.
<i>filename</i>	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 System

xPCFSRMDIR

Remove folder from target computer

Prototype

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

Arguments

<i>port</i>	Enter the value returned by the function xPCOpenTcpIpPort.
<i>dirname</i>	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 System

xPCFSScGetFilename

Get name of file for scope

Prototype

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

Arguments

<i>port</i>	Enter the value returned by the function xPCOpenTcpIpPort.
<i>scNum</i>	Enter the scope number.
<i>filename</i>	The name of the file for the specified scope is stored in <i>filename</i> .

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

xPCFSScSetFilename | Real-Time File Scope

xPCFSScGetWriteMode

Get write mode of file for scope

Prototype

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

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	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

`xPCFSScSetWriteMode` | Real-Time File Scope

xPCFSScGetWriteSize

Get block write size of data chunks

Prototype

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

Arguments

<i>port</i>	Enter the value returned by the function xPCOpenTcpIpPort.
<i>scNum</i>	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

xPCFSScSetWriteSize | Real-Time File Scope

xPCFSScSetFilename

Specify name for file to contain signal data

Prototype

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

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>filename</i>	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.

A fully qualified file name can have a maximum of 260 characters. The file part can have 12 characters—a maximum of eight characters for the file name, one character for the period, and a maximum of three characters for the file extension. If the file name is longer than eight characters, the software truncates it to six characters and adds '~1' to the end of the file name.

See Also

`xPCFSScGetFilename` | Real-Time File Scope

xPCFSScSetWriteMode

Specify when file allocation table entry is updated

Prototype

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

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>writeMode</i>	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

`xPCFSScGetWriteMode` | Real-Time File Scope

xPCFSScSetWriteSize

Specify that memory buffer collect data in multiples of write size

Prototype

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

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>writeSize</i>	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

`xPCFSScGetWriteSize` | Real-Time File Scope

xPCFSWriteFile

Write to file on target computer

Prototype

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

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>fileHandle</i>	Enter the file handle of an open file on the target computer.
<i>numbytes</i>	Enter the number of bytes this function is to write into the file.
<i>data</i>	The contents to write to <i>fileHandle</i> are stored in <i>data</i> .

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

`xPCFSCloseFile` | `xPCFSGetFileSize` | `xPCFSOpenFile` | `xPCFSReadFile`

xPCGetAPIVersion

Get version number of Simulink Real-Time API

Prototype

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

Return

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

Description

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

See Also

`xPCGetTargetVersion`

xPCGetAppName

Return real-time application name

Prototype

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

Arguments

port Enter the value returned by the function xPCOpenTcpIpPort.

model_name The xPCGetAppName function copies the real-time application name character 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 name character string, allocate a buffer of size 256 bytes.

Return

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

Description

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

[xPCIsAppRunning](#) | Real-Time Application Properties

xPCGetEcho

Return display mode for target message window

Prototype

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

Arguments

port Enter the value returned by the function `xPCOpenTcpIpPort`.

Return

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

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

Return

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

Description

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

See Also

`xPCSetEcho`

xPCGetExecTime

Return real-time application execution time

Prototype

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

Arguments

port Enter the value returned by the function `xPCOpenTcpIpPort`.

Return

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

Description

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

See Also

`xPCSetStopTime` | `xPCGetStopTime` | Real-Time Application

xPCGetLastError

Return constant of last error

Prototype

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

Return

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

Description

The `xPCGetLastError` function returns the constant of the last reported error by another API function. This value is reset every time you call a new function. Therefore, you should check this constant value immediately after a call to an API function. For a list of error constants and messages, see “C API Error Messages” on page 1-8.

See Also

`xPCErrorMsg` | `xPCSetLastError`

xPCGetLoadTimeOut

Return timeout value for communication between development and target computers

Prototype

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

Arguments

port Enter the value returned by the function xPCOpenTcpIpPort.

Return

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

Description

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

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

xPCLoadApp | xPCSetLoadTimeOut | xPCUnloadApp

More About

- “Communications Timeout”

xPCGetLogMode

Return logging mode and increment value for real-time application

Prototype

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

Arguments

port Enter the value returned by the function xPCOpenTcpIpPort.

Return

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

Description

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

See Also

xPCSetLogMode | `lgmode`

xPCGetNumOutputs

Return number of outputs

Prototype

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

Arguments

port Enter the value returned by the function xPCOpenTcpIpPort.

Return

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

Description

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

See Also

xPCGetOutputLog | xPCGetNumStates | xPCGetStateLog

xPCGetNumParams

Return number of tunable parameters

Prototype

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

Arguments

port Enter the value returned by the function xPCOpenTcpIpPort.

Return

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

Description

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

See Also

xPCGetParamIdx | xPCSetParam | xPCGetParam | xPCGetParamName |
xPCGetParamDims | Real-Time Application

xPCGetNumScopes

Return number of scopes added to real-time application

Prototype

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

Arguments

port Enter the value returned by the function xPCOpenTcpIpPort.

Return

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

Description

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

xPCGetNumScSignals

Returns number of signals added to specific scope

Prototype

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

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>scopeId</i>	Enter the ID number of the scope for which you want to get the number of added signals.

Return

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

Description

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

xPCGetNumSignals

Return number of signals

Prototype

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

Arguments

port Enter the value returned by the function xPCOpenTcpIpPort.

Return

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

Description

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

See Also

xPCGetSignalIdx | xPCGetSignal | xPCGetSignals | xPCGetSignalName | xPCGetSignalWidth | Real-Time Application

xPCGetNumStates

Return number of states

Prototype

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

Arguments

port Enter the value returned by the function xPCOpenTcpIpPort.

Return

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

Description

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

See Also

xPCGetStateLog | xPCGetNumOutputs | xPCGetOutputLog | Real-Time Application

xPCGetOutputLog

Copy output log data to array

Prototype

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

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>first_sample</i>	Enter the index of the first sample to copy.
<i>num_samples</i>	Enter the number of samples to copy from the output log.
<i>decimation</i>	Select whether to copy every sample value or every Nth value.
<i>output_id</i>	Enter an output identification number.
<i>output_data</i>	The log is stored in <i>output_data</i> , 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 Nth value.

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

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

See Also

`xPCNumLogWraps` | `xPCNumLogSamples` | `xPCMaxLogSamples` | `xPCGetNumOutputs`
| `xPCGetStateLog` | `xPCGetTETLog` | `xPCGetTimeLog` | Real-Time Application

xPCGetParam

Get parameter value and copy it to array

Prototype

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

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>paramIndex</i>	Enter the index for a parameter.
<i>paramValue</i>	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

`xPCSetParam` | `xPCGetParamDims` | `xPCGetParamIdx` | `xPCGetNumParams` | Real-Time Application

xPCGetParamDims

Get row and column dimensions of parameter

Prototype

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

Arguments

<i>port</i>	Enter the value returned by the function xPCOpenTcpIpPort.
<i>paramIndex</i>	Parameter index.
<i>dimension</i>	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

xPCGetParam | xPCGetParamName | xPCGetParamDims | xPCGetParamIdx |
xPCGetNumParams | xPCSetParam | Real-Time Application

xPCGetParamIdx

Return parameter index

Prototype

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

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>blockName</i>	Enter the full block path generated by Simulink Coder™.
<i>paramName</i>	Enter the parameter name for a parameter associated with the block.

Return

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

Description

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

See Also

[xPCGetParam](#) | [xPCGetParamName](#) | [xPCGetParamDims](#) | [Real-Time Application](#)

xPCGetParamName

Get name of parameter

Prototype

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

Arguments

<i>port</i>	Enter the value returned by the function xPCOpenTcpIpPort.
<i>paramIdx</i>	Enter a parameter index.
<i>blockName</i>	Character string with the full block path generated by Simulink Coder.
<i>paramName</i>	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 character strings are unchanged. Get the parameter index from the function xPCGetParamIdx.

See Also

xPCGetParam | xPCGetParamDims | xPCGetParamIdx | xPCGetNumParams |
xPCSetParam | Real-Time Application

xPCGetSampleTime

Return real-time application sample time

Prototype

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

Arguments

port Enter the value returned by the function `xPCOpenTcpIpPort`.

Return

The `xPCGetSampleTime` function returns the sample time, in seconds, of the real-time application. If the function detects an error, it returns `-1`.

Description

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

See Also

`xPCSetSampleTime` | Real-Time Application

xPCGetScope

Get and copy scope data to structure

Prototype

```
scopedata xPCGetScope(int port, int scNum);
```

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	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 `xPCScGetScopePropertyName` 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

`xPCSetScope` | `scopedata` | Real-Time Application

xPCGetScopeList

Get and copy list of scope numbers

Prototype

```
void xPCGetScopeList(int port, int *data);
```

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>data</i>	List of scope numbers in an integer array (allocated by the caller) as a list of unsorted integers.

Description

The `xPCGetScopeList` function gets the list of scopes currently defined. *data* must be large enough to hold the list of scopes. You can query the size by calling the function `xPCGetNumScopes`.

Note: Use the `xPCGetScopeList` function instead of the `xPCGetScopes` function. The `xPCGetScopes` will be removed in a future release.

xPCGetScopes

Get and copy list of scope numbers

Prototype

```
void xPCGetScopes(int port, int *data);
```

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>data</i>	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

`xPCSetScope` | `xPCGetScope` | `xPCScGetSignals` | Real-Time Application

xPCGetSessionTime

Return length of time Simulink Real-Time kernel has been running

Prototype

```
double xPCGetSessionTime(int port);
```

Arguments

port Enter the value returned by the function xPCOpenTcpIpPort.

Return

The xPCGetSessionTime function returns the amount of time in seconds that the Simulink Real-Time kernel has been running on the target computer. If the function detects an error, it returns -1.

Description

The xPCGetSessionTime function returns, as a double, the amount of time in seconds that the Simulink Real-Time kernel has been running. This value is also the time that has elapsed since you last booted the target computer.

xPCGetSignal

Return value of signal

Prototype

```
double xPCGetSignal(int port, int sigNum);
```

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>sigNum</i>	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

`xPCGetSignals` | Real-Time Application

xPCGetSignalIdx

Return index for signal

Prototype

```
int xPCGetSignalIdx(int port, const char *sigName);
```

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>sigName</i>	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 real-time 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 custom program should already know the signal name.

See Also

`xPCGetSignalName` | `xPCGetSignalWidth` | `xPCGetSignal` | `xPCGetSignals` |
Real-Time Application

xPCGetSigIdxfromLabel

Return array of signal indices

Prototype

```
int xPCGetSigIdxfromLabel(int port, const char *sigLabel, int *sigIds);
```

Arguments

<i>port</i>	Enter the value returned by the function xPCOpenTcpIpPort.
<i>sigLabel</i>	Character string with the name of a signal label.
<i>sigIds</i>	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 custom program 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

xPCGetSigLabelWidth | xPCGetSignalLabel

xPCGetSignalLabel

Copy label of signal to character array

Prototype

```
char * xPCGetSignalLabel(int port, int sigIdx, char *sigLabel);
```

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>sigIdx</i>	Enter signal index.
<i>sigLabel</i>	Return signal label associated with signal index, <i>sigIdx</i> .

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 custom program should already know the signal name/label.

See Also

`xPCGetSigIdxfromLabel` | `xPCGetSigLabelWidth`

xPCGetSigLabelWidth

Return number of elements in signal

Prototype

```
int xPCGetSigLabelWidth(int port, const char *sigName);
```

Arguments

port Enter the value returned by the function xPCOpenTcpIpPort.
sigName Character 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 custom program should already know the signal name/label. Signal labels must be unique.

See Also

xPCGetSigIdxfromLabel | xPCGetSignalLabel

xPCGetSignalName

Copy name of signal to character array

Prototype

```
char *xPCGetSignalName(int port, int sigIdx, char *sigName);
```

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>sigIdx</i>	Enter a signal index.
<i>sigName</i>	Character 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

`xPCGetSignalIdx` | `xPCGetSignalWidth` | `xPCGetSignal` | `xPCGetSignals` | Real-Time Application

xPCGetSignals

Return vector of signal values

Prototype

```
int xPCGetSignals(int port, int numSignals, const int *signals,  
double *values);
```

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>numSignals</i>	Enter the number of signals to be acquired (that is, the number of values in <i>signals</i>).
<i>signals</i>	Enter the list of signal numbers to be acquired.
<i>values</i>	Returned values are stored in the double array <i>values</i> .

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`.

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.

See Also

`xPCGetSignal` | `xPCGetSignalIdx`

xPCGetSignalWidth

Return width of signal

Prototype

```
int xPCGetSignalWidth(int port, int sigIdx);
```

Arguments

port Enter the value returned by 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

xPCGetSignalIdx | xPCGetSignalName | xPCGetSignal | xPCGetSignals

xPCGetStateLog

Copy state log values to array

Prototype

```
void xPCGetStateLog(int port, int first_sample, int num_samples,  
int decimation, int state_id, double *state_data);
```

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>first_sample</i>	Enter the index of the first sample to copy.
<i>num_samples</i>	Enter the number of samples to copy from the output log.
<i>decimation</i>	Select whether to copy all the sample values or every Nth value.
<i>state_id</i>	Enter a state identification number.
<i>state_data</i>	The log is stored in <i>state_data</i> , 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 Nth value. For *first_sample*, the sample indices range from 0 to (N-1), where N is the return value of `xPCNumLogSamples`. Use the `xPCNumLogSamples` function to get the maximum number of samples.

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

See Also

`xPCNumLogWraps` | `xPCNumLogSamples` | `xPCMaxLogSamples` | `xPCGetNumStates` | `xPCGetOutputLog` | `xPCGetTETLog` | `xPCGetTimeLog` | Real-Time Application

xPCGetStopTime

Return stop time

Prototype

```
double xPCGetStopTime(int port);
```

Arguments

port Enter the value returned by the function `xPCOpenTcpIpPort`.

Return

The `xPCGetStopTime` function returns the stop time as a double, in seconds, of the real-time application. If the function detects an error, it returns `-10.0`. If the stop time is infinity (run forever), this function returns `-1.0`.

Description

The `xPCGetStopTime` function returns the stop time, in seconds, of the real-time application. This is the amount of time the real-time application runs before stopping. If the function detects an error, it returns `-10.0`. You will then need to use the function `xPCGetLastError` to find the error number.

See Also

`xPCSetStopTime` | Real-Time Application

xPCGetTargetVersion

Get Simulink Real-Time kernel version

Prototype

```
void xPCGetTargetVersion(int port, char *ver);
```

Arguments

port Enter the value returned by the function xPCOpenTcpIpPort.
ver The version is stored in *ver*.

Description

The xPCGetTargetVersion function gets a character string with the version number of the Simulink Real-Time kernel on the target computer. It then copies that version number into *ver*.

See Also

xPCGetAPIVersion

xPCGetTETLog

Copy TET log to array

Prototype

```
void xPCGetTETLog(int port, int first_sample,  
int num_samples, int decimation,  
double *TET_data);
```

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>first_sample</i>	Enter the index of the first sample to copy.
<i>num_samples</i>	Enter the number of samples to copy from the TET log.
<i>decimation</i>	Select whether to copy all the sample values or every Nth value.
<i>TET_data</i>	The log is stored in <i>TET_data</i> , 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 Nth value. For *first_sample*, the sample indices range from 0 to (N-1), where N is the return value of `xPCNumLogSamples`. Use the `xPCNumLogSamples` function to get the maximum number of samples.

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

See Also

`xPCNumLogWraps` | `xPCNumLogSamples` | `xPCMaxLogSamples` | `xPCGetNumOutputs`
| `xPCGetStateLog` | `xPCGetTimeLog` | Real-Time Application

xPCGetTimeLog

Copy time log to array

Prototype

```
void xPCGetTimeLog(int port, int first_sample, int num_samples,  
int decimation, double *time_data);
```

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>first_sample</i>	Enter the index of the first sample to copy.
<i>num_samples</i>	Enter the number of samples to copy from the time log.
<i>decimation</i>	Select whether to copy all the sample values or every Nth value.
<i>time_data</i>	The log is stored in <i>time_data</i> , 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 Nth value. For *first_sample*, the sample indices range from 0 to (N-1), where N is the return value of `xPCNumLogSamples`. Use the `xPCNumLogSamples` function to get the number of samples.

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

See Also

`xPCGetLogMode` | `xPCSetLogMode` | `xPCGetTETLog` | `xPCGetStateLog` |
`xPCMaxLogSamples` | `xPCNumLogSamples` | `xPCNumLogWraps` | Real-Time
Application

xPCInitAPI

Initialize Simulink Real-Time DLL

Prototype

```
int xPCInitAPI(void);
```

Return

The `xPCInitAPI` function returns 0 if it completes execution without detecting an error. If the function detects an error, it returns -1.

Description

The `xPCInitAPI` function initializes the Simulink Real-Time dynamic link library. You must execute this function once at the beginning of the custom program to load the Simulink Real-Time API DLL. This function is defined in the file `xpcinitfree.c`. Link this file with your program.

See Also

`xPCFreeAPI` | `xPCNumLogWraps` | `xPCNumLogSamples` | `xPCMaxLogSamples` | `xPCGetStateLog` | `xPCGetTETLog` | `xPCSetLogMode` | `xPCGetLogMode`

xPCIsAppRunning

Return real-time application running status

Prototype

```
int xPCIsAppRunning(int port);
```

Arguments

port Enter the value returned by the function xPCOpenTcpIpPort.

Return

If the real-time application is stopped, the xPCIsAppRunning function returns 0. If the real-time application is running, this function returns 1. If the function detects an error, it returns -1.

Description

The xPCIsAppRunning function returns 1 or 0 depending on whether the real-time application is stopped or running. If the function detects is an error, use the function xPCGetLastError to check for the error character string constant.

See Also

xPCIsOverloaded | Real-Time Application Properties

xPCIsOverloaded

Return target computer overload status

Prototype

```
int xPCIsOverloaded(int port);
```

Arguments

port Enter the value returned by the function xPCOpenTcpIpPort.

Return

If the real-time application has overloaded the CPU, the `xPCIsOverloaded` function returns 1. If it has not overloaded the CPU, the function returns 0. If this function detects error, it returns -1.

Description

The `xPCIsOverloaded` function checks if the real-time application has overloaded the target computer and returns 1 if it has and 0 if it has not. If the real-time application is not running, the function returns 0.

See Also

`xPCIsAppRunning` | Real-Time Application

xPCIsScFinished

Return data acquisition status for scope

Prototype

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

Arguments

port Enter the value returned by 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

`xPCScGetState` | Real-Time Target Scope | Real-Time File Scope | Real-Time Host Scope

xPCLoadApp

Load real-time application onto target computer

Prototype

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

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>pathstr</i>	Enter the full path to the real-time application file, excluding the file name. For example, in C, use a character string like "C:\\work".
<i>filename</i>	Enter the name of a compiled real-time application (*.dlm) without the file extension. For example, in C use a character string like "xpcosc".

Description

The `xPCLoadApp` function loads the compiled real-time application to the target computer. *pathstr* must not contain the trailing backslash. *pathstr* can be set to NULL or to the character string 'nopath' if the real-time application is in the current folder. The variable *filename* must not contain the real-time application extension.

Before returning, `xPCLoadApp` waits for a certain amount of time before checking whether the model initialization is complete. In the case where the model initialization is incomplete, `xPCLoadApp` returns a timeout error to indicate a connection problem (for example, ETCPREAD). By default, `xPCLoadApp` checks for target readiness five times, with each attempt taking approximately 1 second (less if the target is ready). However, for larger models or models requiring longer initialization (for example, those with thermocouple boards), the default might not be long enough and a spurious timeout can be generated. The functions `xPCGetLoadTimeOut` and `xPCSetLoadTimeOut` control the number of attempts made.

See Also

[xPCGetLoadTimeOut](#) | [xPCSetLoadTimeOut](#) | [xPCUnloadApp](#) | [xPCStopApp](#) | [xPCStartApp](#) | [Real-Time Application](#)

xPCLoadParamSet

Restore parameter values

Prototype

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

Arguments

port Enter the value returned by the function xPCOpenTcpIpPort.
filename Enter the name of the file that contains the saved parameters.

Description

The xPCLoadParamSet function restores the real-time application parameter values saved in the file *filename*. This file must be located on a local drive of the target computer. The parameter file must have been saved from a previous call to xPCSaveParamSet.

See Also

xPCSaveParamSet

xPCMaxLogSamples

Return maximum number of samples that can be in log buffer

Prototype

```
int xPCMaxLogSamples(int port);
```

Arguments

port Enter the value returned by 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

`xPCGetTimeLog` | `xPCGetTETLog` | `xPCGetOutputLog` | `xPCGetStateLog` |
`xPCNumLogWraps` | `xPCNumLogSamples` | Real-Time Application

xPCMaximumTET

Copy maximum task execution time to array

Prototype

```
void xPCMaximumTET(int port, double *data);
```

Arguments

port Enter the value returned by the function xPCOpenTcpIpPort.
data Array of at least two doubles.

Description

The xPCMaximumTET function gets the maximum task execution time (TET) that was achieved during the previous real-time application run. This function also returns the time at which the maximum TET was achieved. The xPCMaximumTET function then copies these values into the *data* array. The maximum TET value is copied into the first element, and the time at which it was achieved is copied into the second element.

Task Execution Time (TET) measures how long it takes the kernel to run for one base-rate time step. For a multirate model, use the profiler to find out what the execution time is for each rate.

See Also

xPCAverageTET | xPCMinimumTET | Real-Time Application

xPCMinimumTET

Copy minimum task execution time to array

Prototype

```
void xPCMinimumTET(int port, double *data);
```

Arguments

port Enter the value returned by the function `xPCOpenTcpIpPort`.
data Array of at least two doubles.

Description

The `xPCMinimumTET` function gets the minimum task execution time (TET) that was achieved during the previous real-time application run. This function also returns the time at which the minimum TET was achieved. The `xPCMinimumTET` function then copies these values into the *data* array. The minimum TET value is copied into the first element, and the time at which it was achieved is copied into the second element.

Task Execution Time (TET) measures how long it takes the kernel to run for one base-rate time step. For a multirate model, use the profiler to find out what the execution time is for each rate.

See Also

`xPCAverageTET` | `xPCMaximumTET` | `xPCIsAppRunning` | Real-Time Application

xPCNumLogSamples

Return number of samples in log buffer

Prototype

```
int xPCNumLogSamples(int port);
```

Arguments

port Enter the value returned by the function `xPCOpenTcpIpPort`.

Return

The `xPCNumLogSamples` function returns the number of samples in the log buffer. If the function detects an error, it returns `-1`.

Description

The `xPCNumLogSamples` function returns the number of samples in the log buffer. In contrast to `xPCMaxLogSamples`, which returns the maximum number of samples that can be logged (because of buffer size constraints), `xPCNumLogSamples` returns the number of samples actually logged.

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

See Also

`xPCGetStateLog` | `xPCGetOutputLog` | `xPCGetTETLog` | `xPCGetTimeLog` | `xPCMaxLogSamples`

xPCNumLogWraps

Return number of times log buffer wraps

Prototype

```
int xPCNumLogWraps(int port);
```

Arguments

port Enter the value returned by 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

`xPCGetTimeLog` | `xPCGetTETLog` | `xPCGetOutputLog` | `xPCGetStateLog` |
`xPCMaxLogSamples` | `xPCNumLogSamples` | Real-Time Application

xPCOpenConnection

Open connection to target computer

Prototype

```
void xPCOpenConnection(int port);
```

Arguments

port Enter the value returned by 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 xPCOpenTcpIpPort can also set up the target information. If the port is already open, calling this function has no effect.

See Also

xPCOpenTcpIpPort | xPCClosePort | xPCReOpenPort | xPCTargetPing |
xPCCloseConnection | xPCRegisterTarget

xPCOpenTcpIpPort

Open TCP/IP connection to Simulink Real-Time system

Prototype

```
int xPCOpenTcpIpPort(const char *ipAddress, const char
*ipPort);
```

Arguments

<i>ipAddress</i>	Enter the IP address of the target as a dotted decimal character string. For example, "192.168.0.10".
<i>ipPort</i>	Enter the associated IP port as a character string. For example, "22222".

Return

The `xPCOpenTcpIpPort` function returns a nonnegative integer that you can then use as the port value for a Simulink Real-Time API function that requires it. If this operation fails, this function returns -1.

Description

The `xPCOpenTcpIpPort` function opens a connection to the TCP/IP location specified by the IP address. It returns a nonnegative integer if it succeeds. Use this integer as the *ipPort* variable in the Simulink Real-Time API functions that require a port value. The global error number is also set, which you can get using `xPCGetLastError`.

See Also

`xPCClosePort` | `xPCReOpenPort` | `xPCTargetPing`

xPCReboot

Reboot target computer

Prototype

```
void xPCReboot(int port);
```

Arguments

port Enter the value returned by the function xPCOpenTcpIpPort.

Description

The xPCReboot function restarts 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 restarted.

See Also

xPCReOpenPort | Real-Time Application

xPCReOpenPort

Reopen communication channel

Prototype

```
int xPCReOpenPort(int port);
```

Arguments

port Enter the value returned by 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 xPCOpenTcpIpPort is that xPCReOpenPort uses the already existing settings, while the other functions need to set up the port.

See Also

xPCOpenTcpIpPort | xPCClosePort

xPCRegisterTarget

Register target with Simulink Real-Time API library

Prototype

```
int xPCRegisterTarget(int commType, const char *ipAddress,  
const char *ipPort, int comPort, int baudRate);
```

Arguments

commType Specify the communication type between the development and target computers. The only value supported is `COMMTYP_TCP_IP`.

Note: RS-232 communication type has been removed. Configure TCP/IP communication instead.

ipAddress Enter the IP address of the target as a dotted decimal character string. For example, "192.168.0.10".

ipPort Enter the associated IP port as a character string. For example, "22222".

Return

When called with TCP/IP parameters, the function returns the port number. If the function detects an error, it returns -1.

When called with RS-232 parameters, the function returns -1 and sets error status `EINVCOMMTYP`.

Description

The `xPCRegisterTarget` function works similarly to `xPCOpenTcpIpPort`, except that it does not try to open a connection to the target computer. In other words, calling

`xPCOpenTcpIpPort` is equivalent to calling `xPCRegisterTarget` with the required parameters, followed by a call to `xPCOpenConnection`.

Use the constant `COMMTYP_TCPIP` for *commType*. The function ignores *comPort* and *baudRate*.

See Also

`xPCDeRegisterTarget` | `xPCOpenTcpIpPort` | `xPCClosePort` | `xPCReOpenPort` | `xPCOpenConnection` | `xPCCloseConnection` | `xPCTargetPing`

xPCRemScope

Remove scope

Prototype

```
void xPCRemScope(int port, int scNum);
```

Arguments

port Enter the value returned by 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

`xPCGetScopes` | `xPCScRemSignal` | `xPCAddScope` | Real-Time Application

xPCSaveParamSet

Save parameter values of real-time application

Prototype

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

Arguments

port Enter the value returned by the function `xPCOpenTcpIpPort`.
filename Enter the name of the file to contain the saved parameters.

Description

The `xPCSaveParamSet` function saves the real-time application parameter values in the file *filename*. This function saves the file on a local drive of the current target computer. You can later reload these parameters with the `xPCLoadParamSet` function.

You might want to save real-time application parameter values if you change these parameter values while the application is running in **Real-Time** mode. Saving these values enable you to easily recreate real-time application parameter values from a number of runs.

See Also

`xPCLoadParamSet`

xPCScAddSignal

Add signal to scope

Prototype

```
void xPCScAddSignal(int port, int scNum, int sigNum);
```

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>sigNum</i>	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

`xPCScRemSignal` | Real-Time Target Scope | Real-Time File Scope | Real-Time Host Scope

xPCScGetAutoRestart

Scope autorestart status

Prototype

```
long xPCScGetAutoRestart(int port, int scNum)
```

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	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

`xPCScSetAutoRestart`

xPCScGetData

Copy scope data to array

Prototype

```
void xPCScGetData(int port, int scNum, int signal_id, int start,  
int numsamples, int decimation, double *data);
```

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>signal_id</i>	Enter a signal number. Enter -1 to get time stamped data.
<i>start</i>	Enter the first sample from which data retrieval is to start.
<i>numsamples</i>	Enter the number of samples retrieved with a decimation of <i>decimation</i> , starting from the <i>start</i> value.
<i>decimation</i>	Enter a value such that every <i>decimation</i> sample is retrieved in a scope window.
<i>data</i>	The data is available in the array <i>data</i> , starting from sample <i>start</i> .

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

[xPCGetScope](#) | [xPCScGetState](#) | [xPCScGetSignals](#) | [xPCScSetDecimation](#) | Real-Time Host Scope

xPCScGetDecimation

Return decimation of scope

Prototype

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

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	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

`xPCScSetDecimation` | Real-Time Host Scope | Real-Time File Scope | Real-Time Target Scope

xPCScGetNumPrePostSamples

Get number of pre- or post-triggering samples before triggering scope

Prototype

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

Arguments

port Enter the value returned by 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

`xPCScSetNumPrePostSamples` | Real-Time Host Scope | Real-Time File Scope | Real-Time Target Scope

xPCScGetNumSamples

Get number of samples in one data acquisition cycle

Prototype

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

Arguments

port Enter the value returned by 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

xPCScSetNumSamples | Real-Time Target Scope | Real-Time File Scope | Real-Time Host Scope

xPCScGetNumSignals

Get number of signals in scope

Prototype

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

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	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

`xPCGetScope` | Real-Time Target Scope | Real-Time File Scope | Real-Time Host Scope

xPCScGetSignalList

Copy list of signals to array

Prototype

```
void xPCScGetSignalList(int port, int scNum, int *data)
```

Arguments

<i>port</i>	Value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>data</i>	Integer array allocated by the caller as a list containing the signal identifiers.

Description

The `xPCScGetSignals` function gets the list of signals defined for scope *scNum*. The array *data* must be large enough to hold the list of signals. To query the size, use the `xPCScGetNumSignals` function. Use the `xPCGetScope` function to get the scope number.

Note: Use the `xPCScGetSignalList` function instead of the `xPCScGetSignals` function. The `xPCScGetSignals` will be removed in a future release.

xPCScGetSignals

Copy list of signals to array

Prototype

```
void xPCScGetSignals(int port, int scNum, int *data);
```

Arguments

<i>port</i>	Value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>data</i>	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

`xPCScGetData` | `xPCGetScopes` | Real-Time File Scope | Real-Time Host Scope | Real-Time Target Scope

xPCScGetStartTime

Get start time for last data acquisition cycle

Prototype

```
double xPCScGetStartTime(int port, int scNum);
```

Arguments

port Enter the value returned by 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

xPCGetScope | Real-Time Target Scope | Real-Time File Scope | Real-Time Host Scope

xPCScGetState

Get state of scope

Prototype

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

Arguments

port Enter the value returned by 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.

Constant	Value	Description
SCST_ACQUIRING	2	Scope is acquiring data. The scope enters this state when it leaves the wait for trigger state.
SCST_FINISHED	3	Scope is finished acquiring data when it has attained the predefined limit.
SCST_INTERRUPTED	4	The user has stopped (interrupted) the scope.

See Also

xPCScStop | xPCScStart | Real-Time File Scope | Real-Time Host Scope | Real-Time Target Scope

xPCScGetTriggerLevel

Get trigger level for scope

Prototype

```
double xPCScGetTriggerLevel(int port, int scNum);
```

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	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

`xPCGetScope` | `xPCScGetTriggerMode` | `xPCScSetTriggerMode` |
`xPCScGetTriggerScope` | `xPCScSetTriggerScope` | `xPCScGetTriggerSignal` |
`xPCScSetTriggerSignal` | `xPCScGetTriggerSlope` | `xPCScSetTriggerSlope` |
`xPCScSetTriggerLevel` | Real-Time File Scope | Real-Time Host Scope | Real-Time Target Scope

xPCScGetTriggerMode

Get trigger mode for scope

Prototype

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

Arguments

port Enter the value returned by the function xPCOpenTcpIpPort.
scNum Enter the scope number.

Return

The xPCScGetTriggerMode function returns the scope trigger mode. If the function detects an error, it returns -1.

Description

The xPCScGetTriggerMode function gets the trigger mode for scope *scNum*. Use the xPCGetScope function to get the scope number. Use the constants defined in `xpcapiconst.h` to interpret the trigger mode. These constants include the following:

Constant	Value	Description
TRIGMD_FREERUN	0	There is no trigger mode. The scope triggers when it is ready to trigger, regardless of the circumstances.
TRIGMD_SOFTWARE	1	Only user intervention can trigger the scope. No other triggering is possible.
TRIGMD_SIGNAL	2	The scope is triggered only after a signal has crossed a value.

Constant	Value	Description
TRIGMD_SCOPE	3	The scope is triggered by another scope at the trigger point of the triggering scope, modified by the value of <code>triggerscopesample</code> (see <code>scopedata</code>).

See Also

`xPCScSetTriggerMode` | `xPCScGetTriggerScope` | `xPCScSetTriggerScope` | `xPCScGetTriggerSignal` | `xPCScSetTriggerSignal` | `xPCScGetTriggerSlope` | `xPCScSetTriggerSlope` | `xPCScGetTriggerLevel` | `xPCScSetTriggerLevel` | `xPCGetScope` | Real-Time File Scope | Real-Time Host Scope | Real-Time Target Scope

xPCScGetTriggerScope

Get trigger scope

Prototype

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

Arguments

port Enter the value returned by 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

xPCScGetTriggerMode | xPCScSetTriggerMode | xPCScGetTriggerSignal |
xPCScSetTriggerSignal | xPCScGetTriggerSlope | xPCScSetTriggerSlope
| xPCScGetTriggerLevel | xPCScSetTriggerLevel | xPCGetScope | Real-Time
Host Scope | Real-Time File Scope | Real-Time Target Scope

xPCScGetTriggerScopeSample

Get sample number for triggering scope

Prototype

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

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	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

`xPCScSetTriggerScopeSample` | `xPCScGetTriggerMode` | `xPCScSetTriggerMode` | `xPCScGetTriggerScope` | `xPCScSetTriggerScope` | `xPCScGetTriggerSignal` | `xPCScSetTriggerSignal` | `xPCScGetTriggerSlope` | `xPCScSetTriggerSlope` | `xPCScGetTriggerLevel` | `xPCScSetTriggerLevel` | `xPCGetScope` | Real-Time Host Scope | Real-Time File Scope | Real-Time Target Scope

xPCScGetTriggerSignal

Get trigger signal for scope

Prototype

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

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	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

`xPCScGetTriggerMode` | `xPCScSetTriggerMode` | `xPCScGetTriggerScope` | `xPCScSetTriggerScope` | `xPCScSetTriggerSignal` | `xPCScGetTriggerSlope` | `xPCScSetTriggerSlope` | `xPCScGetTriggerLevel` | `xPCScSetTriggerLevel` | `xPCGetScope` | Real-Time Host Scope | Real-Time File Scope | Real-Time Target Scope

xPCScGetTriggerSlope

Get trigger slope for scope

Prototype

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

Arguments

port Enter the value returned by 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

xPCScGetTriggerMode | xPCScSetTriggerMode | xPCScGetTriggerScope |
xPCScSetTriggerScope | xPCScGetTriggerSignal | xPCScSetTriggerSignal
| xPCScSetTriggerSlope | xPCScGetTriggerLevel | xPCScSetTriggerLevel |
xPCGetScope | Real-Time Host Scope | Real-Time File Scope | Real-Time Target Scope

xPCScGetType

Get type of scope

Prototype

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

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.

Return

The `xPCScGetType` function returns the scope type. If the function detects an error, it returns -1.

Description

The `xPCScGetType` function gets the type (`SCTYPE_HOST` for host, `SCTYPE_TARGET` for target, or `SCTYPE_FILE` for file) of scope *scNum*. Use the constants defined in `xpcapiconst.h` to interpret the return value. A scope of type `SCTYPE_HOST` is displayed on the development computer while a scope of type `SCTYPE_TARGET` is displayed on the target computer screen. A scope of type `SCTYPE_FILE` is stored on a storage medium. Use the `xPCGetScope` function to get the scope number.

See Also

`xPCGetScope` | Real-Time Target Scope | Real-Time File Scope | Real-Time Host Scope

xPCScRemSignal

Remove signal from scope

Prototype

```
void xPCScRemSignal(int port, int scNum, int sigNum);
```

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>sigNum</i>	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

`xPCScGetState` | `xPCScGetSignals` | `xPCGetScopes` | `xPCRemScope` | `xPCAddScope` | `xPCScAddSignal` | `xPCGetScope` | Real-Time Host Scope | Real-Time File Scope | Real-Time Target Scope

xPCScSetAutoRestart

Scope autorestart status

Prototype

```
void xPCScSetAutoRestart(int port, int scNum, int autorestart)
```

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>autorestart</i>	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

`xPCScGetAutoRestart` | Real-Time Target Scope | Real-Time File Scope | Real-Time Host Scope

xPCScSetDecimation

Set decimation of scope

Prototype

```
void xPCScSetDecimation(int port, int scNum, int decimation);
```

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>decimation</i>	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

`xPCScGetState` | `xPCScGetDecimation` | `xPCGetScope` | Real-Time File Scope | Real-Time Host Scope | Real-Time Target Scope

xPCScSetNumPrePostSamples

Set number of pre- or posttriggering samples before triggering scope

Prototype

```
void xPCScSetNumPrePostSamples(int port, int scNum, int prepost);
```

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>prepost</i>	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

`xPCScGetState` | `xPCScGetNumPrePostSamples` | `xPCGetScope` | Real-Time File Scope | Real-Time Host Scope | Real-Time Target Scope

xPCScSetNumSamples

Set number of samples in one data acquisition cycle

Prototype

```
void xPCScSetNumSamples(int port, int scNum, int samples);
```

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>samples</i>	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.

For file scopes, the `NumSamples` parameter works with the `autorestart` parameter.

- Autorestart is on — When the scope triggers, the scope collects data into a memory buffer. A background task examines the buffer and writes data to the disk continuously, appending new data to the end of the file. When the scope reaches the number of samples that you specified, it starts collecting data again, overwriting the memory buffer. If the background task cannot keep pace with data collection, data can be lost.
- Autorestart is off — When the scope triggers, the scope collects data into a memory buffer up to the number of samples that you specified, and then stops. A background task examines the buffer and writes data to the disk continuously, appending the new data to the end of the file.

See Also

`xPCScGetState` | `xPCScGetNumSamples` | `xPCGetScope` | Real-Time File Scope | Real-Time Host Scope | Real-Time Target Scope

xPCScSetTriggerLevel

Set trigger level for scope

Prototype

```
void xPCScSetTriggerLevel(int port, int scNum, double level);
```

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>level</i>	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

`xPCScGetTriggerSlope` | `xPCScSetTriggerSignal` | `xPCScGetTriggerSignal` | `xPCScSetTriggerScope` | `xPCScGetTriggerScope` | `xPCScSetTriggerMode` | `xPCScGetTriggerMode` | `xPCScGetState` | `xPCScSetTriggerSlope` | `xPCScGetTriggerLevel` | `xPCGetScope` | Real-Time Host Scope | Real-Time File Scope | Real-Time Target Scope

xPCScSetTriggerMode

Set trigger mode of scope

Prototype

```
void xPCScSetTriggerMode(int port, int scNum, int mode);
```

Arguments

port Enter the value returned by 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 <code>triggerscopesample</code> (see <code>scopedata</code>).

See Also

xPCGetScopes | xPCScSetTriggerLevel | xPCScGetTriggerLevel |
xPCScSetTriggerSlope | xPCScGetTriggerSlope | xPCScSetTriggerSignal |
xPCScGetTriggerSignal | xPCScSetTriggerScope | xPCScGetTriggerScope |
xPCScGetTriggerMode | xPCScGetState | xPCGetScope | Real-Time Host Scope |
Real-Time File Scope | Real-Time Target Scope

xPCScSetTriggerScope

Select scope to trigger another scope

Prototype

```
void xPCScSetTriggerScope(int port, int scNum, int trigScope);
```

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>trigScope</i>	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

`xPCGetScopes` | `xPCScSetTriggerLevel` | `xPCScGetTriggerLevel` | `xPCScSetTriggerSlope` | `xPCScGetTriggerSlope` | `xPCScSetTriggerSignal` | `xPCScGetTriggerSignal` | `xPCScGetTriggerScope` | `xPCScSetTriggerMode` | `xPCScGetTriggerMode` | `xPCScGetState` | `xPCGetScope` | Real-Time Host Scope | Real-Time File Scope | Real-Time Target Scope

xPCScSetTriggerScopeSample

Set sample number for triggering scope

Prototype

```
void xPCScSetTriggerScopeSample(int port, int scNum, int trigScSamp);
```

Arguments

port Enter the value returned by 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

`xPCGetScopes` | `xPCScSetTriggerLevel` | `xPCScGetTriggerLevel` |
`xPCScSetTriggerSlope` | `xPCScGetTriggerSlope` | `xPCScSetTriggerSignal` |
`xPCScGetTriggerSignal` | `xPCScSetTriggerScope` | `xPCScGetTriggerScope` |
`xPCScSetTriggerMode` | `xPCScGetTriggerMode` | `xPCScGetTriggerScopeSample`

| xPCGetScope | Real-Time File Scope | Real-Time Host Scope | Real-Time Target
Scope

xPCScSetTriggerSignal

Select signal to trigger scope

Prototype

```
void xPCScSetTriggerSignal(int port, int scNum, int trigSig);
```

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>trigSig</i>	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

`xPCGetScopes` | `xPCScGetState` | `xPCScSetTriggerLevel` | `xPCScGetTriggerLevel` | `xPCScSetTriggerSlope` | `xPCScGetTriggerSlope` | `xPCScGetTriggerSignal` | `xPCScSetTriggerScope` | `xPCScGetTriggerScope` | `xPCScSetTriggerMode` | `xPCScGetTriggerMode` | `xPCGetScope` | Real-Time Host Scope | Real-Time File Scope | Real-Time Target Scope

xPCScSetTriggerSlope

Set slope of signal that triggers scope

Prototype

```
void xPCScSetTriggerSlope(int port, int scNum, int trigSlope);
```

Arguments

port Enter the value returned by 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

`xPCGetScopes` | `xPCScSetTriggerLevel` | `xPCScGetTriggerLevel` |
`xPCScGetTriggerSlope` | `xPCScSetTriggerSignal` | `xPCScGetTriggerSignal`

| xPCScSetTriggerScope | xPCScGetTriggerScope | xPCScSetTriggerMode |
xPCScGetTriggerMode | xPCScGetState | xPCGetScope | Real-Time Host Scope |
Real-Time File Scope | Real-Time Target Scope

xPCScSoftwareTrigger

Set software trigger of scope

Prototype

```
void xPCScSoftwareTrigger(int port, int scNum);
```

Arguments

port Enter the value returned by 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

`xPCGetScopes` | `xPCScGetState` | `xPCIsScFinished` | `xPCGetScope` | Real-Time Host Scope | Real-Time File Scope | Real-Time Target Scope

xPCScStart

Start data acquisition for scope

Prototype

```
void xPCScStart(int port, int scNum);
```

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	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

`xPCGetScopes` | `xPCScGetState` | `xPCScStop` | `xPCIsScFinished` | `xPCGetScope`
| `Real-Time File Scope` | `Real-Time Host Scope` | `Real-Time Target Scope`

xPCScStop

Stop data acquisition for scope

Prototype

```
void xPCScStop(int port, int scNum);
```

Arguments

port Enter the value returned by 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

`xPCGetScopes` | `xPCScStart` | `xPCScGetState` | `xPCGetScope` | Real-Time Host Scope | Real-Time File Scope | Real-Time Target Scope

xPCSetEcho

Turn message display on or off

Prototype

```
void xPCSetEcho(int port, int mode);
```

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>mode</i>	Valid values are
0	Turns the display off
1	Turns the display on

Description

On the target computer screen, the `xPCSetEcho` function sets the message display on the target computer on or off. You can change the mode only when the real-time application is stopped. When you turn the message display off, the message screen no longer updates. Existing messages remain on the screen as they were.

See Also

`xPCGetEcho`

xPCSetLastError

Set last error to specific character string constant

Prototype

```
void xPCSetLastError(int error);
```

Arguments

error Specify the character 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 character string constant to ENOERR, indicating no error was found.

See Also

xPCGetLastError | xPCErrorMsg

xPCSetLoadTimeOut

Change initialization timeout value between development and target computers

Prototype

```
void xPCSetLoadTimeOut(int port, int timeOut);
```

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>timeOut</i>	Enter the new communication timeout value.

Description

The `xPCSetLoadTimeOut` function changes the timeout value for communication between the development and target computers. The *timeOut* value is the time a Simulink Real-Time API function waits for the communication to complete before returning. It enables you to set the number of communication attempts to be made before signaling a timeout.

For example, the function `xPCLoadApp` waits to check whether the model initialization for a new real-time application is complete before returning. When a new real-time application is loaded onto the target computer, the function `xPCLoadApp` waits for a certain time to check whether the model initialization is complete before returning. If the model initialization is incomplete within the allotted time, `xPCLoadApp` returns a timeout error.

By default, `xPCLoadApp` checks for target readiness for up to 5 seconds. However, for larger models or models requiring longer initialization (for example, models with thermocouple boards), the default might not be long enough and a spurious timeout can be generated. Other functions that communicate with the target computer will wait for *timeOut* seconds before declaring a timeout event.

See Also

`xPCLoadApp` | `xPCGetLoadTimeOut` | `xPCUnloadApp` | Real-Time Application

xPCSetLogMode

Set logging mode and increment value of scope

Prototype

```
void xPCSetLogMode(int port, lgmode logging_data);
```

Arguments

<i>port</i>	Enter the value returned by the function xPCOpenTcpIpPort.
<i>logging_data</i>	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

lgmode | xPCGetLogMode | Real-Time Application

xPCSetParam

Change value of parameter

Prototype

```
void xPCSetParam(int port, int paramIdx, const double *paramValue);
```

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>paramIdx</i>	Parameter index.
<i>paramValue</i>	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

`xPCGetParamDims` | `xPCGetParamIdx` | `xPCGetParam`

xPCSetSampleTime

Change real-time application sample time

Prototype

```
void xPCSetSampleTime(int port, double ts);
```

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>ts</i>	Sample time for the real-time application.

Description

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

Note: Some blocks produce incorrect results when you change their sample time at run time. If you include such blocks in your model, the software displays a warning message during model build. To avoid incorrect results, change the sample time in the original model, and then rebuild and download the model.

See Also

`xPCGetSampleTime` | Real-Time Application

xPCSetScope

Set properties of scope

Prototype

```
void xPCSetScope(int port, scopedata state);
```

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>state</i>	Enter a structure of type <code>scopedata</code> .

Description

Note: The `xPCSetScope` function will be removed in a future release. Use the `xPCScSetScopePropertyName` 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

`xPCGetScope` | `xPCScGetState` | `scopedata`

xPCSetStopTime

Change real-time application stop time

Prototype

```
void xPCSetStopTime(int port, double tfinal);
```

Arguments

port Enter the value returned by the function xPCOpenTcpIpPort.
tfinal Enter the stop time, in seconds.

Description

The xPCSetStopTime function sets the stop time of the real-time application to the value in *tfinal*. The real-time application will run for this number of seconds before stopping. Set *tfinal* to `-1.0` to set the stop time to infinity.

See Also

xPCGetStopTime | Real-Time Application

xPCStartApp

Start real-time application

Prototype

```
void xPCStartApp(int port);
```

Arguments

port Enter the value returned by the function `xPCOpenTcpIpPort`.

Description

The `xPCStartApp` function starts the real-time application loaded on the target computer.

See Also

`xPCStopApp` | Real-Time Application

xPCStopApp

Stop real-time application

Prototype

```
void xPCStopApp(int port);
```

Arguments

port Enter the value returned by the function xPCOpenTcpIpPort.

Description

The xPCStopApp function stops the real-time application loaded on the target computer. The real-time application remains loaded and the parameter changes you made remain intact. If you want to stop and unload an application, use xPCUnloadApp.

See Also

xPCUnloadApp | xPCStartApp | Real-Time Application

xPCTargetPing

Ping target computer

Prototype

```
int xPCTargetPing(int port);
```

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
-------------	--

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 character 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.

Note that `xPCTargetPing` will cause the target computer to close the TCP/IP connection. You can use `xPCOpenConnection` to reconnect. You can also use this `xPCTargetPing` feature to close the target computer connection in the event of an aborted TCP/IP connection (for example, if the program running on your development computer has a fatal error).

See Also

`xPCOpenConnection` | `xPCOpenTcpIpPort` | `xPCClosePort`

xPCTgScGetGrid

Get status of grid line for particular scope

Prototype

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

Arguments

port Enter the value returned by 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

xPCGetScopes | xPCTgScSetGrid | xPCTgScSetViewMode | xPCTgScGetViewMode
| xPCTgScSetMode | xPCTgScGetMode | xPCTgScSetYLimits |
xPCTgScGetYLimits | Real-Time Target Scope

xPCTgScGetMode

Get scope mode for displaying signals

Prototype

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

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	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 of scope *scNum*, which must be of type `SCTYPE_TARGET`. The mode is one of `SCMODE_NUMERICAL`, `SCMODE_REDRAW`, and `SCMODE_SLIDING`, `SCMODE_ROLLING`. Use the `xPCGetScopes` function to get a list of scopes.

See Also

xPCGetScopes | xPCTgScSetGrid | xPCTgScGetGrid | xPCTgScSetViewMode
| xPCTgScGetViewMode | xPCTgScSetMode | xPCTgScSetYLimits |
xPCTgScGetYLimits | Real-Time Target Scope

xPCTgScGetViewMode

Get view mode for target computer display

Prototype

```
int xPCTgScGetViewMode(int port);
```

Arguments

port Enter the value returned by the function xPCOpenTcpIpPort.

Return

0.

Description

Note: xPCTgScGetViewMode has no function. It returns 0.

See Also

xPCGetScopes | xPCTgScSetGrid | xPCTgScGetGrid | xPCTgScSetViewMode |
xPCTgScSetMode | xPCTgScGetMode | xPCTgScSetYLimits | xPCTgScGetYLimits
| Real-Time Target Scope

xPCTgScGetYLimits

Copy y-axis limits for scope to array

Prototype

```
void xPCTgScGetYLimits(int port, int scNum, double *limits);
```

Arguments

<i>port</i>	Enter the value returned by the function xPCOpenTcpIpPort.
<i>scNum</i>	Enter the scope number.
<i>limits</i>	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

xPCGetScopes | xPCTgScSetGrid | xPCTgScGetGrid | xPCTgScSetViewMode
| xPCTgScGetViewMode | xPCTgScSetMode | xPCTgScGetMode |
xPCTgScSetYLimits | Real-Time Target Scope

xPCTgScSetGrid

Set grid mode for scope

Prototype

```
void xPCTgScSetGrid(int port, int scNum, int grid);
```

Arguments

<i>port</i>	Enter the value returned by the function xPCOpenTcpIpPort.
<i>scNum</i>	Enter the scope number.
<i>grid</i>	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

xPCGetScopes | xPCTgScGetGrid | xPCTgScSetViewMode | xPCTgScGetViewMode
| xPCTgScSetMode | xPCTgScGetMode | xPCTgScSetYLimits |
xPCTgScGetYLimits | Real-Time Target Scope

xPCTgScSetMode

Set display mode for scope

Prototype

```
void xPCTgScSetMode(int port, int scNum, int mode);
```

Arguments

<i>port</i>	Enter the value returned by the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>mode</i>	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

`xPCGetScopes` | `xPCTgScSetGrid` | `xPCTgScGetGrid` | `xPCTgScSetViewMode`
| `xPCTgScGetViewMode` | `xPCTgScGetMode` | `xPCTgScSetYLimits` |
`xPCTgScGetYLimits` | Real-Time Target Scope

xPCTgScSetViewMode

Set view mode for scope

Prototype

```
void xPCTgScSetViewMode(int port, int scNum);
```

Arguments

<i>port</i>	Enter the value returned by the function xPCOpenTcpIpPort.
<i>scNum</i>	Enter the scope number.

Description

Note: xPCTgScSetViewMode has no function.

See Also

xPCGetScopes | xPCTgScSetGrid | xPCTgScGetGrid | xPCTgScGetViewMode |
xPCTgScSetMode | xPCTgScGetMode | xPCTgScSetYLimits | xPCTgScGetYLimits
| Real-Time Target Scope

xPCTgScSetYLimits

Set *y*-axis limits for scope

Prototype

```
void xPCTgScSetYLimits(int port, int scNum, const double *Ylimits);
```

Arguments

port Enter the value returned by 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

`xPCGetScopes` | `xPCTgScSetGrid` | `xPCTgScGetGrid` | `xPCTgScSetViewMode`
| `xPCTgScGetViewMode` | `xPCTgScSetMode` | `xPCTgScGetMode` |
`xPCTgScGetYLimits` | Real-Time Target Scope

xPCUnloadApp

Unload real-time application

Prototype

```
void xPCUnloadApp(int port);
```

Arguments

port Enter the value returned by the function xPCOpenTcpIpPort.

Description

The xPCUnloadApp function stops the current real-time application, removes it from the target computer memory, and resets the target computer in preparation for receiving a new real-time application. The function xPCLoadApp calls this function before loading a new real-time application.

See Also

xPCLoadApp | Real-Time Application

MATLAB API

fc422mexcalcbits

Calculate parameter values for Fastcom 422/2-PCI board

Syntax

```
[a,b] = fc422mexcalcbits(frequency)
[a,b,df] = fc422mexcalcbits(frequency)
```

Description

`[a,b] = fc422mexcalcbits(frequency)` accepts a baud and converts this value into a vector containing values for the parameter **Clocks Bits** of the Fastcom[®] 422/2-PCI driver clock. These values set the phase-locked-loop parameters for the board.

`[a,b,df] = fc422mexcalcbits(frequency)` accepts a baud and converts this value into a vector containing the phase-locked-loop parameters for the board and the resulting baud value.

Examples

Clocks Bits Values

```
[a,b] = fc422mexcalcbits(30e3)
```

```
a =
```

```
2111792
```

```
b =
```

```
23
```

In the RS-232/RS-422/RS-485 Send/Receive (Composite) block parameters, **Board Setup** tab, set **Clock Bits** to [2111792 23].

Clocks Bits Values with Actual Result

```
[a,b,df] = fc422mexcalcbits(1.49e6)
```

```
a =
```

```
3805896
```

```
b =
```

```
23
```

```
df =
```

```
1.4901e+06
```

In the RS-232/RS-422/RS-485 Send/Receive (Composite) block parameters, **Board Setup** tab, set **Clock Bits** to [3805896 23].

Input Arguments

frequency — Baud for the board, in symbols/second

positive-valued scalar

The baud must be between 30e3 and 1.5e6. This limitation is a physical limitation of the clock circuit.

Example: 30e3

Data Types: double

Output Arguments

[a, b] — Values for driver block parameter

vector of scalars

a, b – Values for the driver block parameter. These values set the phase-locked-loop parameters for the board.

[a, b, df] – Values for driver block parameter and resulting baud value

vector of scalars

- **a, b** – Values for the driver block parameter. These values set the phase-locked-loop parameters for the board.
- **df** – The actual baud value that the driver block parameter creates. The clock circuit has limited resolution and is unable to match an arbitrary frequency perfectly.

See Also

RS-232/RS-422/RS-485 Send/Receive (Composite)

Introduced in R2014a

macaddr

Convert character vector-based MAC address to vector-based address

Syntax

```
macaddr(MAC_address)
```

Description

`macaddr(MAC_address)` converts a character vector-based MAC address to a vector-based MAC address.

Examples

Simple

```
macaddr('01:23:45:67:89:ab')
ans =
     1    35    69   103   137   171
```

Input Arguments

MAC_address — MAC address to be converted

delimited character vector

The value is entered as a character vector comprised of six colon-delimited fields of two-digit hexadecimal numbers.

Example: '01:23:45:67:89:ab'

Data Types: char

See Also

“Model-Based Ethernet Communications”

Introduced in R2014a

profile_slrt

Collect profiling data

Syntax

```
profData = profile_slrt(profileInfo)
```

Description

`profData = profile_slrt(profileInfo)` collects and displays execution profiling data from a target computer that is running a suitably configured real-time application. By default, it displays an execution profile plot and a code execution profiling report.

Task Execution Time (TET) measures how long it takes the kernel to run for one base-rate time step. For a multirate model, use the profiler to find out what the execution time is for each rate.

To configure a model for execution profiling, check the **Measure task execution time** option in the **Verification** tab of the Code Generation dialog box. If you also want to profile function execution times, select the **Measure function execution times** check box.

After setting these options, you must build, download, and run the real-time application before calling `profile_slrt`.

Examples

Concurrent Execution Example

Profile the concurrent execution model `dxpcmds6t` using default settings on a multicore target computer.

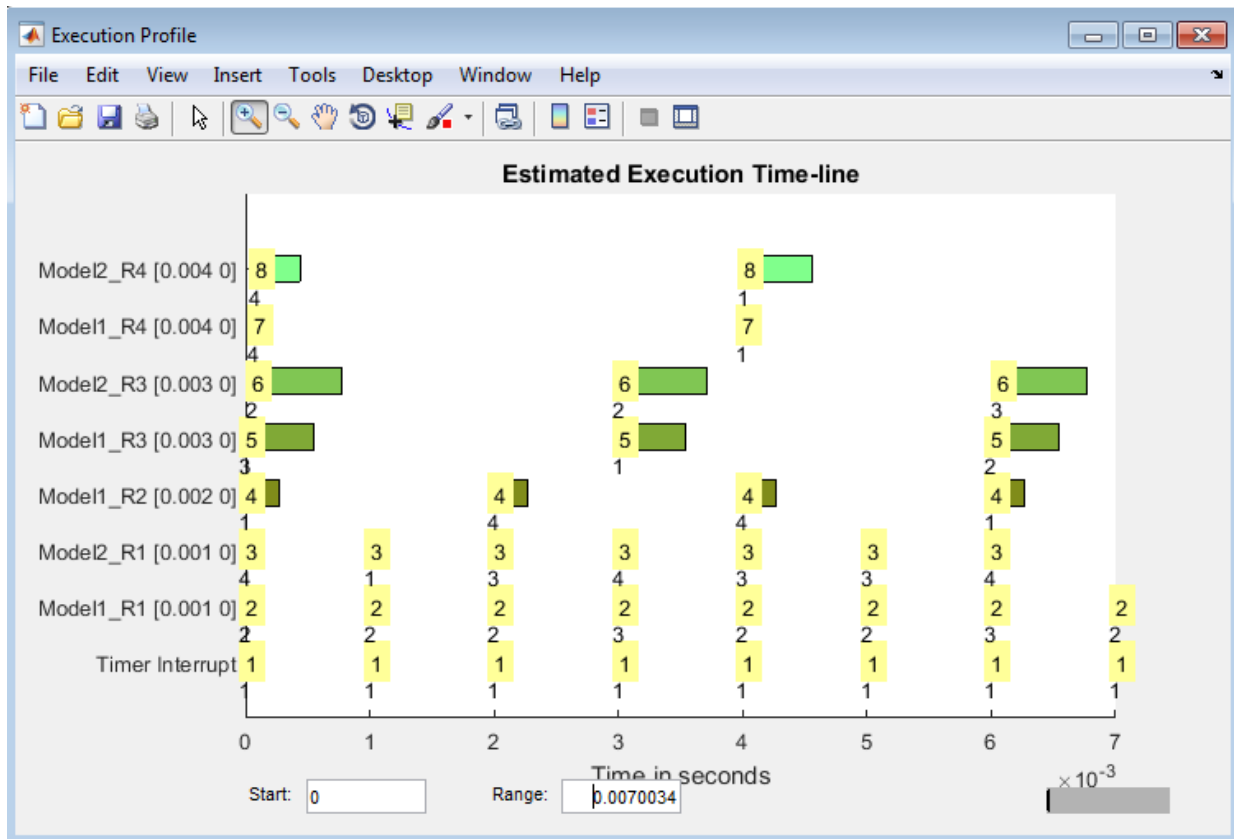
Configure model `dxpcmds6t` for profiling.

Build, download, and execute the model.

Profile the real-time application execution.

```
profileInfo.modelname = 'dxpcmds6t.mdl';
profData = profile_slrt(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)	433413090
Measured time display options	('Units', 'Seconds', 'ScaleFactor', '1e-09', 'NumericFormat', '%0.0f')
Timer frequency (ticks per second)	3.49207e+09
Profiling data created	22-Jun-2016 17:34:49


2. Profiled Sections of Code

Section	Maximum Turnaround Time	Average Turnaround Time	Maximum Execution Time	Average Execution Time	Calls	
Timer Interrupt	3434	1268	3434	1268	536	
Model1_R1 [0.001 0]	55860	53871	55860	53871	535	
Model2_R1 [0.001 0]	64347	63324	64347	63324	535	
Model1_R2 [0.002 0]	268426	267493	268426	267493	268	
Model1_R3 [0.003 0]	541506	537853	541506	537853	179	
Model2_R3 [0.003 0]	717207	714600	717207	714600	179	
Model1_R4 [0.004 0]	8149	7430	8149	7430	134	
Model2_R4 [0.004 0]	548031	545981	548031	545981	134	

OK Help

Profile Data	Description
Maximum turnaround time	Longest time between when the task starts and finishes. This time includes task preemptions (interrupts).

Profile Data	Description
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** button  in the Coder 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 development or target computer

0 (default) | 1

- 0 — The raw data file `xPCTrace.csv` is on the target computer. Transfer the file from the target computer to the host.
- 1 — The raw data file `xPCTrace.csv` is in the current folder on the development computer.

Data Types: double

modelName — Name of the model to be profiled

username

The name can include the model file extension.

Data Types: char

noPlot — Flag suppressing execution profile plot

0 (default) | 1

- 0 — Display the execution profile plot on the development computer monitor.
- 1 — Do not display the execution profile plot on the development computer monitor.

Data Types: double

noreport — Flag suppressing code execution profiling report

0 (default) | 1

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

Data Types: double

Output Arguments

profData — Profile results data

structure

Profile results data stored in an object of type `coder.profile.ExecutionTime`.

TimerTicksPerSecond — Number of seconds per timer tick

double

Scales the execution time tick.

Sections — Array of results data for profiled code sections

array

Each array item is an object of type `coder.profile.ExecutionTimeSection`.

More About

- “Execution Profiling for Real-Time Applications”
- “Failure to Read Profiling Data”

See Also

Sections | TimerTicksPerSecond

Introduced in R2014a

slrt

Create object that manages target computer

Syntax

```
target_object = slrt
target_object = slrt(target_name)
```

Description

`target_object = slrt` constructs a target object representing the default target computer.

When MATLAB evaluates the return value on the development computer, it attempts to connect to the target computer. If the attempt succeeds, MATLAB prints **Connected = Yes**, followed by the status of the real-time application running on the target computer. If the attempt fails, MATLAB waits until the connection times out, and then prints **Connected = No**. To avoid the timeout delay, verify that the target computer is operational and connected to the development computer, or suppress output with a terminating semicolon.

`target_object = slrt(target_name)` constructs a target object representing the target computer designated by `target_name`.

Examples

Default Target Computer

Create a target object that communicates with the default target computer. Report the status of the default target computer. In this case, the target computer is connected to the development computer and is executing the loader.

```
target_object = slrt
```

```
Target: TargetPC1
Connected          = Yes
```

```
Application = loader
```

Specific Target Computer

Create a target object that communicates with target computer `TargetPC1`. Report the status of the target computer. In this case, the target computer is not connected to the development computer.

```
target_object = slrt('TargetPC1')
```

```
Target: TargetPC1
Connected = No
```

Input Arguments

target_name — Name assigned to target computer

character vector

Example: 'TargetPC1'

Data Types: char

Output Arguments

target_object — Object representing target computer

`SimulinkRealTime.target` object

Object that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required Ethernet link settings.

Example: `tg`

Data Types: struct

See Also

`SimulinkRealTime.target` | Target Settings Properties

Introduced in R2014a

slrtbench

Benchmark Simulink Real-Time models on target computer

Syntax

```
slrtbench
slrtbench benchmark
slrtbench benchmark -reboot
slrtbench benchmark -cleanup
slrtbench benchmark -verbose
slrtbench benchmark -reboot -cleanup -verbose
```

```
expected_results = slrtbench()
current_results = slrtbench(benchmark, ___)
```

Description

`slrtbench` 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 target 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.

Benchmark execution includes generating benchmark models, building and downloading real-time applications, searching for the minimal achievable sample time, and displaying results.

Note: In R2017a, function `slrtbench` will be removed. Use `SimulinkRealTime.utils.minimumSampleTime` or `Performance Advisor` instead.

Depending upon the value of `benchmark`, `slrtbench benchmark` produces different outputs:

- `slrtbench this` displays benchmark results for your target computer, compared with the representative benchmark results for other target computers:
 - Relative Performance — Bar graph containing the target 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` displays the following information:
 - Benchmark name
 - Number of blocks
 - Model build time in seconds
 - Execution time in seconds
 - Minimal achievable sample time in microseconds.

`slrtbench benchmark -reboot` runs the benchmark, then restarts the target computer.

`slrtbench benchmark -cleanup` runs the benchmark, plots or prints benchmark results, and deletes the build files.

`slrtbench benchmark -verbose` prints build output, runs the benchmark, and plots or prints benchmark results.

`slrtbench benchmark -reboot -cleanup -verbose` prints build output, restarts the target computer, deletes build files, and plots or prints results.

You can add zero or more of these control arguments in arbitrary order.

`expected_results = slrtbench()` returns the benchmark results for the five predefined benchmarks in a structure array.

Depending upon the value of `benchmark`, `current_results = slrtbench(benchmark, ___)` returns different results:

- `slrtbench('this')` returns the benchmark results for the predefined benchmarks in a structure array.
- `slrtbench(benchmark)` returns the benchmark results for the specified model in a structure.

Examples

`slrtbench`

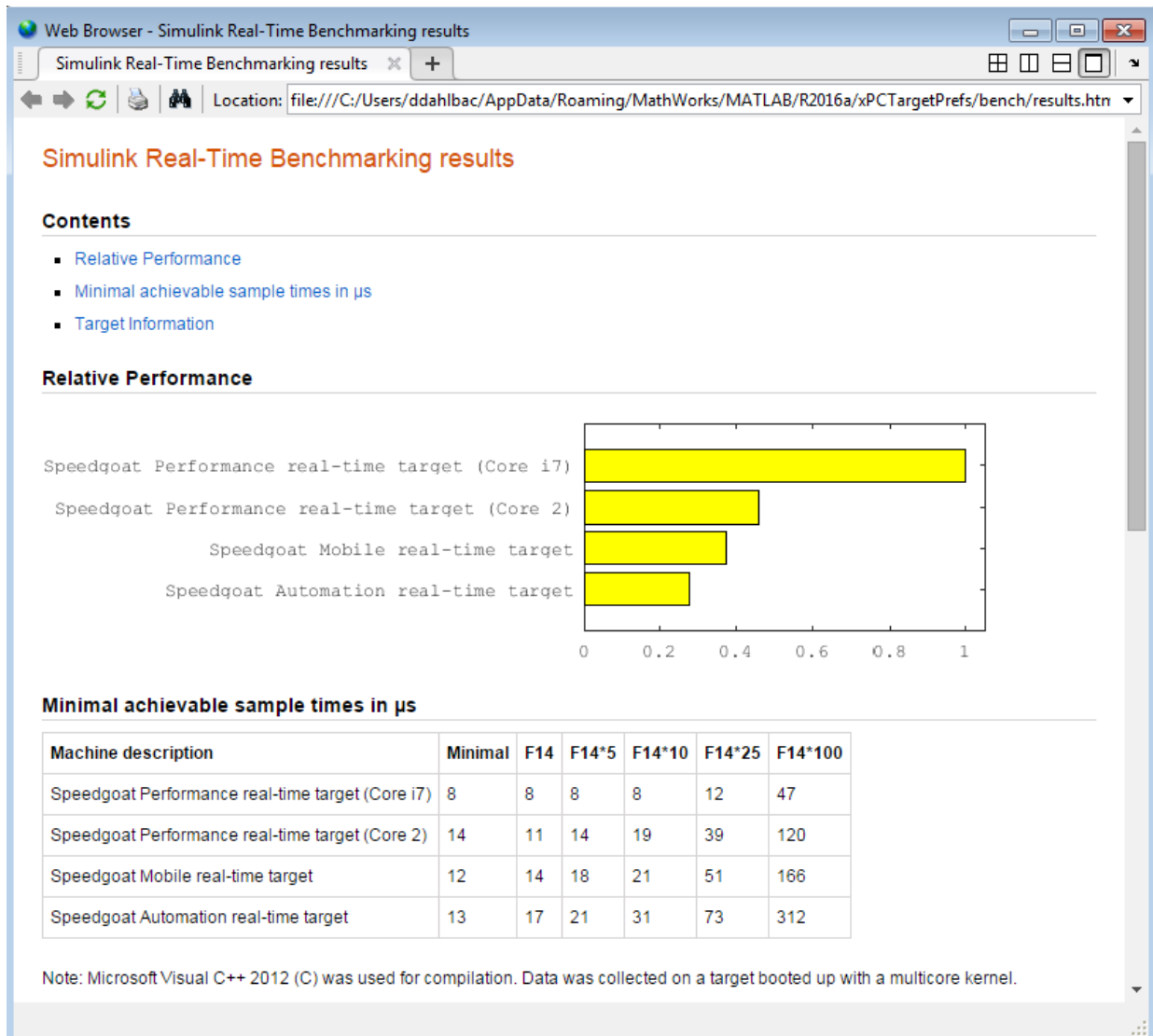
Show representative benchmark results from various target computers.

Start the target computer and run confidence test.

```
slrttest
```

Display representative results on predefined benchmarks.

```
slrtbench
```



slrtbench this

Benchmark the target computer with the predefined benchmarks.

Start the target computer and run confidence test.

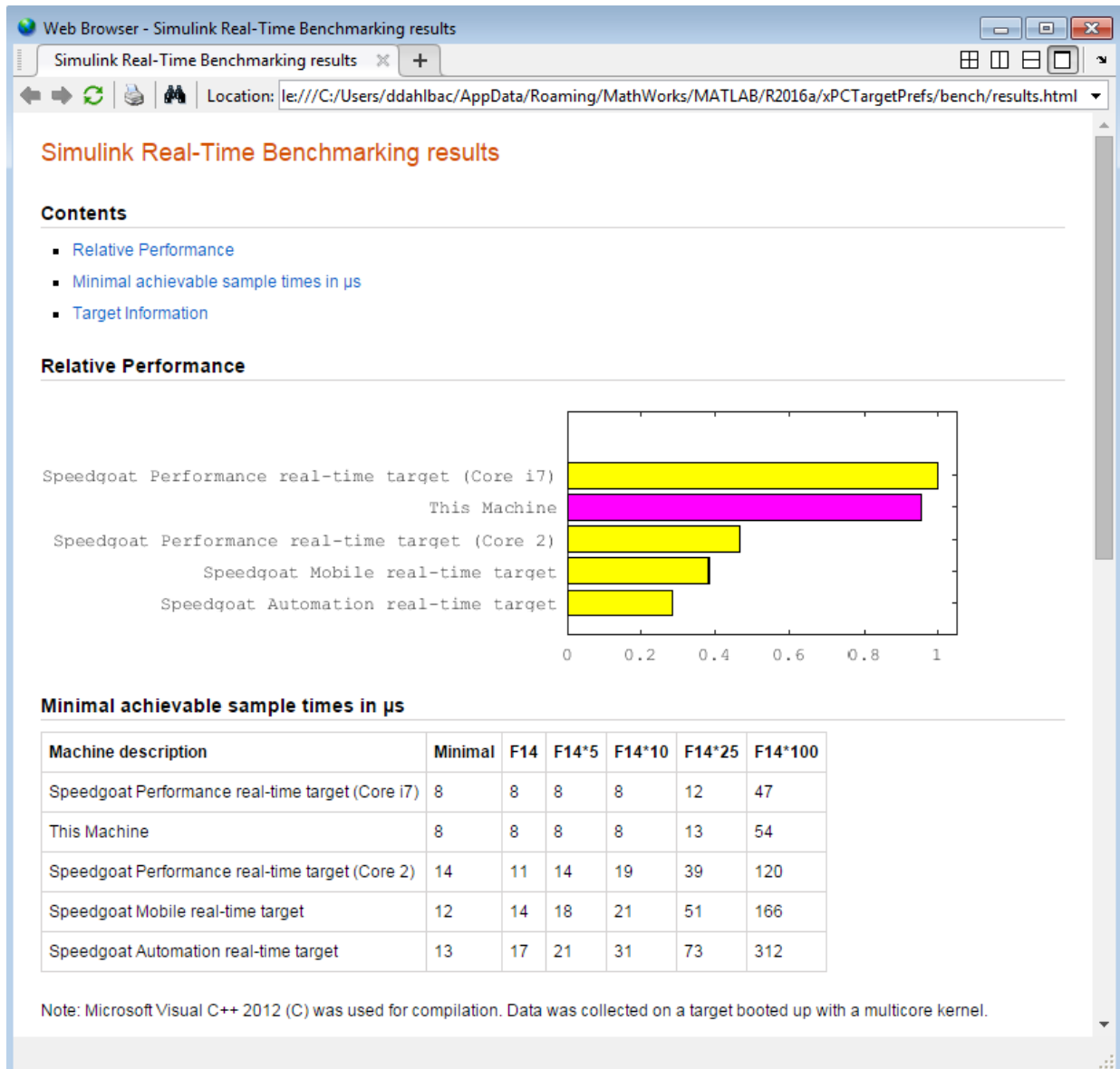
```
slrttest
```

Run the benchmark models and display results.

```
slrtbench this
```

```
Warning: SLRTBENCH will be removed in a future release. Use  
SimulinkRealTime.utils.minimumSampleTime instead.
```

```
### 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
```



`slrtbench this -verbose -reboot -cleanup`

Benchmark the target computer with the predefined benchmarks and all control options.

Start the target computer and run confidence test.

`slrttest`

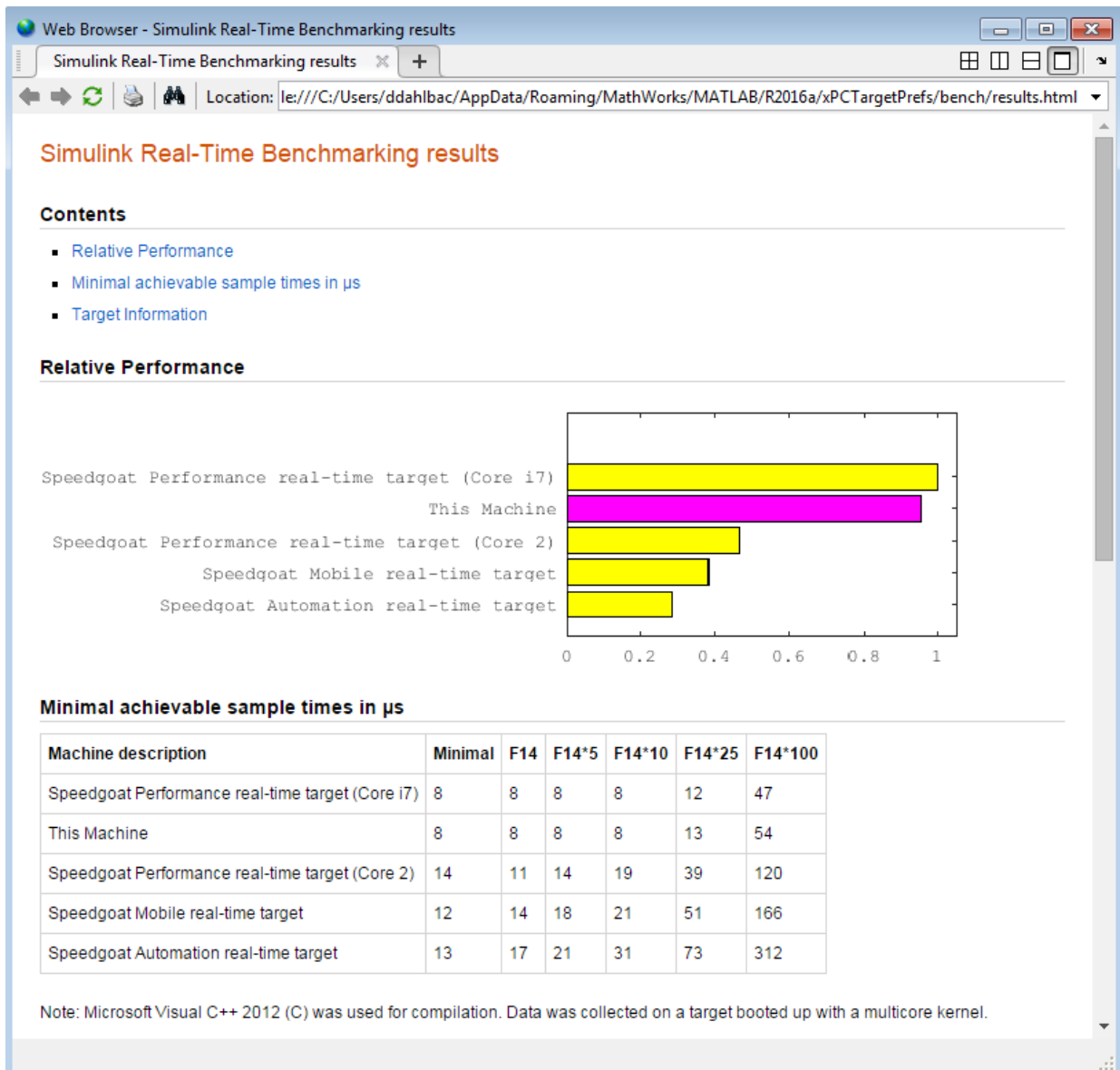
Run the benchmark models, restart the target computer, delete build files, and display results.

`slrtbench this -verbose -reboot -cleanup`

```
Warning: SLRTBENCH will be removed in a future release. Use
SimulinkRealTime.utils.minimumSampleTime instead.
```

```
### 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.
```



slrtbench xpcosc

Use model xpcosc to benchmark the target computer, and then clean up build files.

Start the target computer and run confidence test.

```
slrttest
```

Run benchmark on `xpcosc`, delete build files, and print results.

```
slrtbench xpcosc
```

```
Warning: SLRTBENCH will be removed in a future release. Use  
SimulinkRealTime.utils.minimumSampleTime instead.
```

```
### 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):   13.5  
Elapsed time for model benchmark (sec): 45.3  
Minimal achievable sample time (microsec): 8.5
```

```
slrtbench xpcosc --verbose -reboot -cleanup
```

Use model `xpcosc` to benchmark the target computer with all control options.

Start the target computer and run confidence test.

```
slrttest
```

Run benchmark on `xpcosc`, restart the target computer, delete build files, and print results.

```
slrtbench xpcosc -verbose -reboot -cleanup
```

```
Warning: SLRTBENCH will be removed in a future release. Use  
SimulinkRealTime.utils.minimumSampleTime instead.
```

```
### 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):    20.0
Elapsed time for model benchmark (sec): 45.3
Minimal achievable sample time (microsec): 8.5

```

expected_results = slrtbench()

Return a structure array containing benchmark results showing what to expect of various target computers.

Start the target computer and run confidence test.

slrttest

Return an array with representative results for each processor type, in arbitrary order.

```

expected_results = slrtbench();
expected_results(1)

```

```

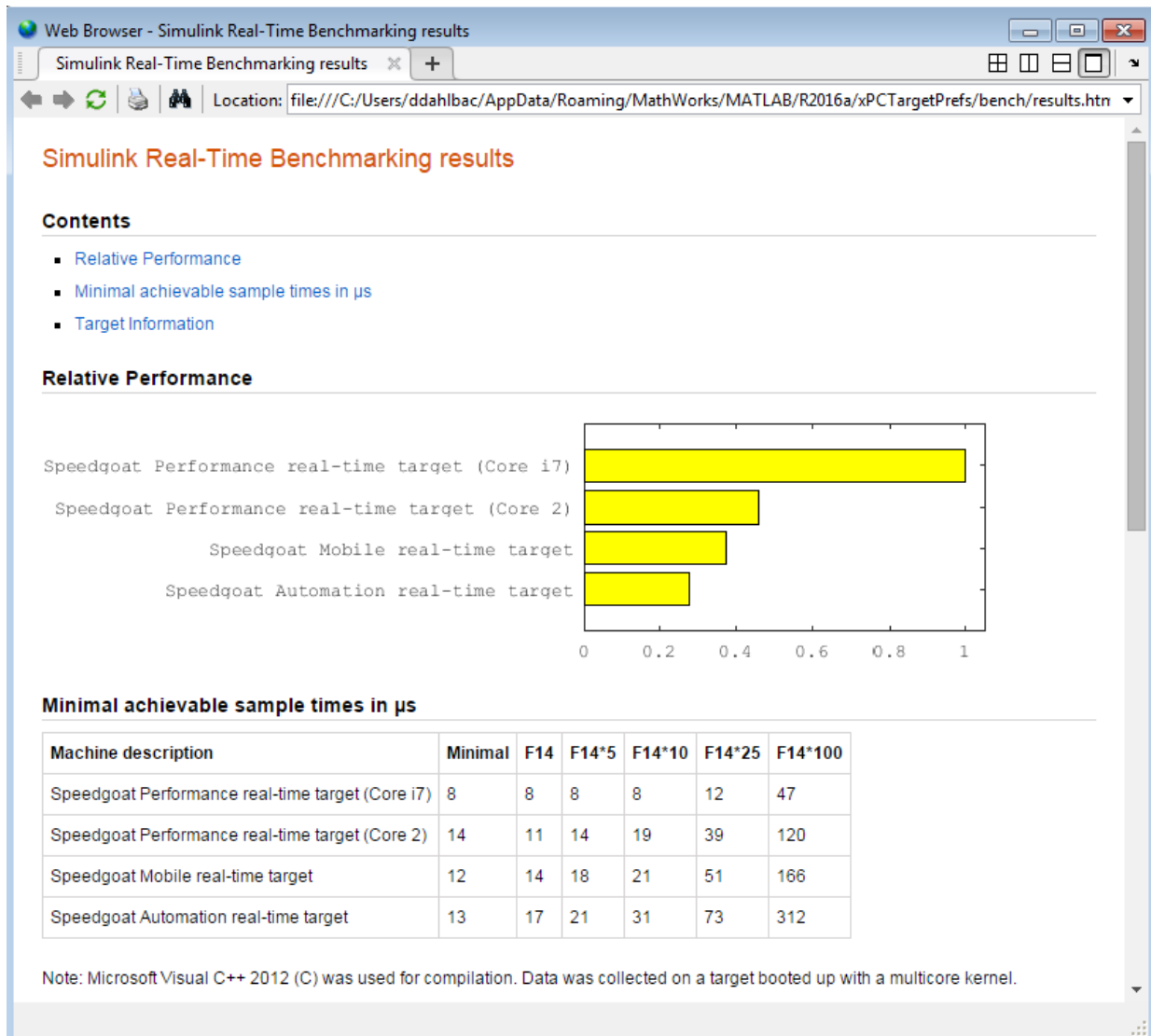
ans =

```

```

    Machine: 'Speedgoat Performance real-time target (Core i7)'
BenchResults: [1x6 double]
    Desc: '% Intel(R) Core(TM) i7-3770K CPU @ 3.50GHz...'

```



```
current_results = slrtbench('xpcosc', '-verbose', '-reboot', '-cleanup')
```

Benchmark the target computer using the `xpcosc` model with all control options. Return a structure array with results.

Start the target computer and run confidence test.

```
slrttest
```

Build 'xpcosc', print build messages, run benchmark, restart the target computer, delete build files, and return results.

```
current_results = slrtbench('xpcosc', '-verbose', '-reboot',
    '-cleanup')
```

```
Warning: SLRTBENCH will be removed in a future release. Use
SimulinkRealTime.utils.minimumSampleTime instead.
```

```
### 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: 14.5824
    BenchTime: 45.2125
```

Tsmin: 8.4844e-06

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 character vector or character vector variable containing one of:

<code>this</code>	All five predefined benchmark models (minimal, f14, f14*5, f14*10, and f14*25)
<code>usermdl</code>	Your model, <i>usermdl</i> .
<code>minimal</code>	Minimal model consisting of three blocks (Constant, Gain, Termination).
<code>f14</code>	Standard Simulink example f14 (62 blocks, 10 continuous states).
<code>f14*5</code>	Five f14 systems modeled in subsystems (310 blocks, 50 continuous states).
<code>f14*10</code>	Ten f14 systems (620 blocks, 100 continuous states).
<code>f14*25</code>	25 f14 systems (1550 blocks, 250 continuous states).
<code>f14*100</code>	100 f14 systems (6200 blocks, 1000continuous states).

When using function form, enclose literal arguments in single quotes.

Example: 'this'

Example: '-reboot'

Data Types: char

Output Arguments

expected_results — Results of predefined benchmarks previously run on representative target computers

struct array

Contains representative benchmark results in a structure array with element fields:

<i>Machine</i>	Target computer information character vector containing CPU type, CPU speed, compiler
<i>BenchResults</i>	Target computer benchmark performance for all five predefined benchmarks
<i>Desc</i>	Target computer descriptor character vector containing machine type, RAM size, cache size

current_results — Current results of specified benchmark

struct

Contains actual benchmark results in a structure with fields:

<i>Name</i>	Benchmark name
<i>nBlocks</i>	Number of blocks in benchmark
<i>BuildTime</i>	Elapsed time in seconds to build benchmark
<i>BenchTime</i>	Elapsed time in seconds to run benchmark
<i>Tsmin</i>	Minimal achievable sample time in seconds for benchmark

More About

Tips

- Before you run `slrtbench`, you must be able to do the following:
 - Start the target computer.

- Connect the development computer to the target computer.
- 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` does not have continuous or discrete states. It provides an indication of the target computer interrupt latencies.
- www.mathworks.com/support/compilers/current_release

See Also

`slrttest`

Introduced in R2014a

slrtdrivertool

Construct skeleton for custom driver

Syntax

```
slrtdrivertool
```

Description

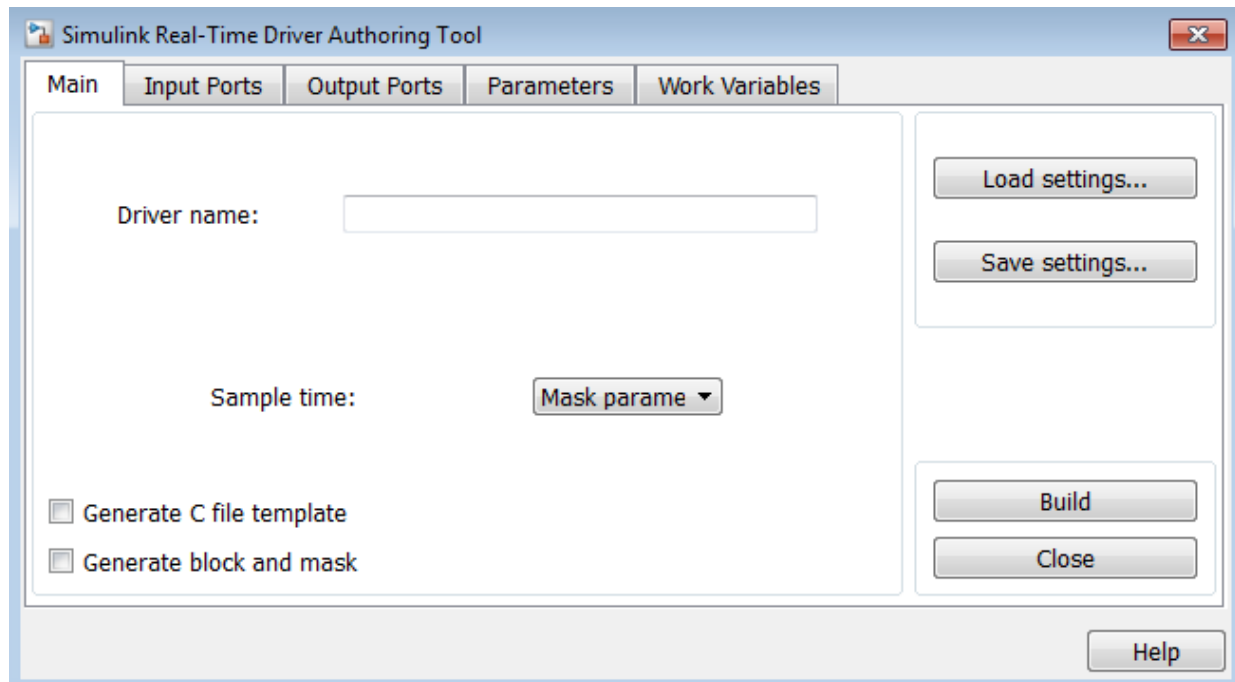
slrtdrivertool opens the Simulink Real-Time Driver Authoring Tool. Using this tool, you can:

- Define the driver name.
- Specify how the sample time is defined (inherited or as a mask parameter).
- Define input and output ports.
- Define parameters and working variables.
- Generate a C file template (optional).
- Generate a block and mask dialog box (optional).
- Save and load settings.
- Build a skeleton driver.

Examples

Define a skeleton driver

```
slrtdrivertool
```



Introduced in R2014a

slrtexplr

Configure target computer and real-time application for execution

Syntax

```
slrtexplr
```

Description

Typing `slrtexplr` at the MATLAB command prompt opens Simulink Real-Time Explorer.

From within Simulink Real-Time Explorer, you can export a session as a standalone executable that runs without MATLAB.

When you run Simulink Real-Time Explorer from within MATLAB, you have available the full capabilities of Simulink Real-Time Explorer. When you run it as a standalone executable, you have available a subset of the capabilities of Simulink Real-Time Explorer.

- Environment configuration
 - Configure and view communication parameters.
 - Configure target computer settings
 - Configure target computer startup
 - Browse target computer file system.
- Control
 - Load, run, and unload real-time applications on the target computer.
 - Connect to and disconnect from the target computer.
 - Change stop time and sample times without regenerating code.
 - Record task execution time during or after last run.
- Instrumentation

- Create graphical instrument panels for acquiring signals and tuning parameters.
- Save and load instrument panels.
- Start and stop instrument panels.
- Use instrument panels to interact with real-time applications.
- Signal acquisition
 - Create, save, and load signal groups.
 - Monitor signals.
 - Add and configure host, target, or file scopes.
 - Attach signals to or remove signals from scopes.
 - Start and stop scopes.
 - Attach signals to instruments.
- Parameter tuning
 - Create, save, and load parameter groups.
 - Display and tune parameters.
 - Attach parameters to instruments.
- Window configuration
 - Make multiple workspaces visible simultaneously.
 - Move workspaces around the window.
 - Export model configuration as a standalone executable.
 - Save and restore model configuration layouts.

When you run Simulink Real-Time Explorer as a standalone executable, it has the following restrictions:

- You cannot change the communication parameters that the interface uses to communicate with the target computer. Before you export the Simulink Real-Time Explorer configuration, configure and test the communication parameters.

To access more than one target computer, in the **Targets** window, configure a separate **Session** record for each target computer.

- For each instrument, the exporting software records the real-time application and target computer environment with which it is associated. To interact with multiple

target computers, create separate instrument panels for each separate real-time application and target computer combination.

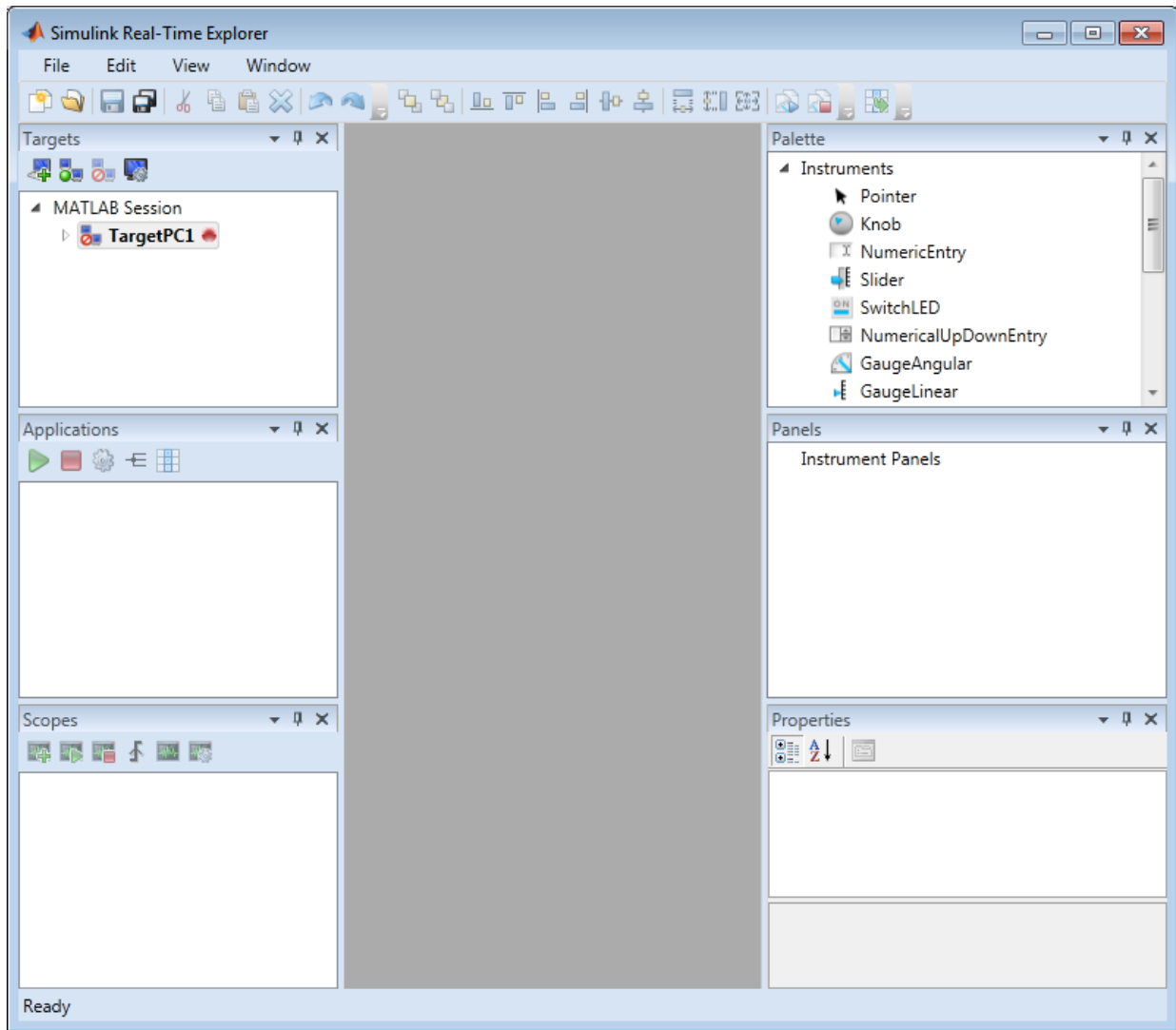
- If you rename a target computer, update the **TargetName** parameter for each associated instrument to maintain the connection to the real-time application.
- You cannot load or unload a real-time application from the standalone executable. Before you start the executable, start the real-time application on the target computer.
- You can access only instrument panels and windows that you loaded before you exported the configuration.
- You cannot access the real-time application model hierarchy from the standalone executable.
- You can access only signals in signal groups that you loaded before you exported the configuration.
- You cannot move a signal from one signal group to another group, or create or load a new signal group.
- You can access only parameters in parameter groups that you loaded before you exported the configuration.
- You cannot move a parameter from one parameter group to another group, or create or load a new parameter group.
- You cannot save session layouts. If you close a window, you can restore the original layout using **File > Restore Original View**.

Examples

Default

Open Simulink Real-Time Explorer

```
slrtexplr
```



More About

- “PCI Bus Ethernet Setup”
- “USB-to-Ethernet Setup”

- “Target Computer Settings”
- “Target Boot Methods”
- “Execute Real-Time Application with Simulink Real-Time Explorer”
- “Monitor Signals with Simulink Real-Time Explorer”
- “Create Target Scopes with Simulink Real-Time Explorer”
- “Create Host Scopes with Simulink Real-Time Explorer”
- “Create File Scopes with Simulink Real-Time Explorer”
- “Tune Parameters with Simulink Real-Time Explorer”
- “Instrumentation for Real-Time Applications”
- “Explorer Configuration Exported to Run Outside MATLAB”
- “Guidelines for Exporting Explorer Configuration”

Introduced in R2014a

slrtgetCC

Compiler settings for development computer environment

Syntax

```
slrtgetCC
type = slrtgetCC
type = slrtgetCC('Type')
location = slrtgetCC('Location')
[type,location] = slrtgetCC
slrtgetCC('supported')
slrtgetCC('installed')
[compilers] = slrtgetCC('installed')
```

Description

`slrtgetCC` displays the compiler type and location in the Command Window.

`type = slrtgetCC` and `type = slrtgetCC('Type')` both return the compiler type in `type`.

`location = slrtgetCC('Location')` returns the compiler location in `location`.

The `mex -setup` command sets the default compiler for Simulink Real-Time builds, provided the MEX compiler is a supported Microsoft compiler. `slrtgetCC` returns the result of the `slrtsetCC` command only, not the result of the `mex` command. If `slrtgetCC` returns an empty character vector as `location`, Simulink Real-Time is using the MEX compiler.

`[type,location] = slrtgetCC` returns the compiler type and its location in `type` and `location`.

`slrtgetCC('supported')` displays the compiler versions supported by the Simulink Real-Time environment.

`slrtgetCC('installed')` displays the supported compilers installed on the development computer.

[compilers] = slrtgetCC('installed') returns in a structure the supported compilers installed on the development computer.

Examples

Display compiler type and location

```
slrtgetCC
```

```
Compiler Settings:
```

```
    Type = VisualC  
    Location = C:\Program Files (x86)\Microsoft Visual Studio 10.0
```

Return compiler type

```
type = slrtgetCC('Type')
```

```
type =
```

```
VisualC
```

Return compiler location

```
location = slrtgetCC('Location')
```

```
location =
```

```
C:\Program Files (x86)\Microsoft Visual Studio 10.0
```

Return compiler type and location

```
[type, location] = slrtgetCC
```

```
type =
```

```
VisualC
```

```
location =
```

```
C:\Program Files (x86)\Microsoft Visual Studio 10.0
```

Display supported compilers

```
slrtgetCC('supported')
```

List of C++ Compilers supported by Simulink Real-Time:

Name	Version	Service Packs
Microsoft Visual C++ Compilers 2008	9.0	1
Microsoft Visual C++ Compilers 2010	10.0	1
Microsoft Visual C++ Compilers 2012	11.0	
Microsoft Visual C++ Compilers (Windows SDK) 2010	10.0	1

Display supported compilers installed

```
slrtgetCC('installed')
```

List of installed C++ Compilers:

```
Name: Microsoft Visual C++ Compilers 2008 Professional Edition  
(SP1)
```

```
Location: c:\Program Files (x86)\Microsoft Visual Studio 9.0
```

```
Name: Microsoft Visual C++ Compilers 2010 Professional
```

```
Location: C:\Program Files (x86)\Microsoft Visual Studio 10.0
```

Return supported compilers installed

```
[compilers] = slrtgetCC('installed')  
compilers(1)
```

```
compilers =
```

```
1x2 struct array with fields:
```

```
    Type  
    Name  
    Location
```

```
ans =
```

```
    Type: 'VisualC'  
    Name: 'Microsoft Visual C++ Compilers 2008 Professional  
          Edition (SP1)'
```

Location: 'c:\Program Files (x86)\Microsoft Visual Studio 9.0'

Output Arguments

type — Type of compiler
VisualC

Simulink Real-Time supports the Microsoft Visual Studio C compiler only.

location — Folder path to compiler on development computer
character vector

compilers — Array of structures containing compiler type, name, and location
array of structures

More About

- www.mathworks.com/support/compilers/current_release

See Also

mex | slrtsetCC

Introduced in R2014a

slrtpingtarget

Test communication between development and target computers

Syntax

```
slrtpingtarget  
slrtpingtarget target_computer_name
```

Description

`slrtpingtarget` without an argument returns `success` if the development computer and the default target computer can communicate using the settings for that target computer. Otherwise, it returns `failed`.

`slrtpingtarget target_computer_name` returns `success` if the development computer can communicate with target computer `target_computer_name` using the settings for that target computer. Otherwise, it returns `failed`.

Examples

Check communication with default target computer

```
slrtpingtarget
```

Check communication with specified target computer

```
slrtpingtarget TargetPC1
```

Input Arguments

target_computer_name — Name of specific target computer

TargetPC1 | TargetPC2 | ...

Name property of a particular target computer environment object. The default name is TargetPC1.

When using function form, enclose the argument in single quotes ('TargetPC1').

Example: TargetPC1

Data Types: char

Introduced in R2014a

slrtsetCC

Compiler settings for development computer environment

Syntax

```
slrtsetCC setup  
slrtsetCC 'type' 'location'
```

Description

`slrtsetCC setup` queries the development computer for installed C compilers supported by the Simulink Real-Time environment. You can then select the C compiler.

The command `mex -setup` sets the default compiler for Simulink Real-Time builds, provided the MEX compiler is a supported Microsoft compiler. Use `slrtsetCC('setup')` only if you must specify different compilers for MEX and Simulink Real-Time.

`slrtsetCC 'type' 'location'` sets the compiler type and location.

To return to the default MEX compiler from a setting by `slrtsetCC`, type `slrtsetCC 'VisualC' ''`, setting the compiler location to the empty character vector.

Examples

Compiler Selection

```
slrtsetCC setup
```

Select your compiler for Simulink Real-Time.

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


[0] None

Compiler:2

Verify your selection:

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

Are these correct [y]/n?y

Done...

Compiler Specification

```
slrtsetCC 'VisualC',
          'C:\Program Files (x86)\Microsoft Visual Studio 10.0'
```

Input Arguments

type — Type of compiler

VisualC (default)

type must be VisualC, representing the Microsoft Visual Studio C compiler.

Example: 'VisualC'

Data Types: char

location — Folder path to compiler on development computer

character vector

Data Types: char

More About

- www.mathworks.com/support/compilers/current_release

See Also

mex | slrtgetCC

Introduced in R2014a

slrttest

Test Simulink Real-Time installation

Syntax

```
slrttest
slrttest noreboot
slrttest target_name, ___
```

Description

slrttest is a confidence test that checks the following tasks:

- Initiate communication between the development and target computers.
- Restart the target computer and reset the target environment.
- Build a real-time application on the development computer.
- Download a real-time application to the target computer.
- Check communication between the development and target computers using commands.
- Execute a real-time application.
- Compare the results of a simulation and the real-time application run.

slrttest noreboot skips the restart test on the default target computer. Use this option if the target computer does not support software restart.

slrttest target_name, ___ runs the tests on the target computer identified by target_name.

Examples

Test Default Target Computer

Target computer must be running and physically connected to the development computer.

slrttest

```
### Simulink Real-Time Test Suite
### Host-Target interface is: TcpIp
### Test 1, Ping target computer 'TargetPC1' using
system ping: OK
### Test 2, Ping target computer 'TargetPC1' using
SLRTPINGTARGET: OK
### Test 3, Software reboot the target computer
'TargetPC1': OK
### Test 4, Build and download a Simulink Real-Time application
using model slrttestmdl to target computer 'TargetPC1': OK
### Test 5, Check host-target command communications with
'TargetPC1': OK
### Test 6, Download a pre-built Simulink Real-Time application
to target computer 'TargetPC1': ... OK
### Test 7, Execute the Simulink Real-Time application
for 0.2s: OK
### Test 8, Upload logged data and compare with simulation
results: OK
### Test Suite successfully finished
```

Test Default Target Computer, Skipping Restart Test

Target computer must be running and physically connected to the development computer.

```
slrttest noreboot
```

Test Specified Target Computer, Skipping Restart Test

Target computer must be running and physically connected to the development computer.

```
slrttest 'TargetPC1' noreboot
```

Input Arguments

target_name — Specifies target name
character vector

The target name character vector is case sensitive.

Example: 'TargetPC1'

More About

- “Troubleshooting in Simulink Real-Time”

Introduced in R2014a

SimulinkRealTime.addTarget

Add new Simulink Real-Time target object

Syntax

```
SimulinkRealTime.addTarget('target_name')
```

Description

`SimulinkRealTime.addTarget('target_name')` adds the definition for a new target computer, represented by the name `'mytarget'`. It returns an object of type `SimulinkRealTime.targetSettings` corresponding to the new target computer.

Examples

Add a Simulink Real-Time target object `'TargetPC2'` to the system:

```
tg = SimulinkRealTime.addTarget('TargetPC2')
```

The `tg` variable contains the attributes of the new target computer.

See Also

`SimulinkRealTime.getTargetSettings` | `SimulinkRealTime.removeTarget`

Introduced in R2014a

SimulinkRealTime.copyFileToHost

Copy file from target computer to development computer

Syntax

```
SimulinkRealTime.copyFileToHost(file_name)
SimulinkRealTime.copyFileToHost(target_obj, file_name)
```

Description

`SimulinkRealTime.copyFileToHost(file_name)` copies file `file_name` from the default target computer to the development computer.

`SimulinkRealTime.copyFileToHost(target_obj, file_name)` copies file `file_name` from the target computer represented by `target_obj` to the development computer.

Examples

Copy File by Name from Default Target Computer

Copy file from current folder on default target computer.

```
SimulinkRealTime.copyFileToHost('data.dat')
```

Copy File by Full Path from Specified Target Computer

Copy file from full path location on target computer `TargetPC1`.

```
tg = slrt('TargetPC1');
SimulinkRealTime.copyFileToHost(tg, 'c:\xpcosc\data1.dat')
```

Input Arguments

target_obj — Name of a target computer or a variable containing a target computer object
character vector | object

If the argument is a character vector, 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 character vector | full path name character vector

If the argument is a file name, the file must be in the current folder on the target computer, as indicated by the function `SimulinkFileSystem.pwd`.

The file is transferred from the target and written with the same file name to the current folder on the development computer.

Example: 'myFile.txt'

Example: 'c:\subDir\myFile.txt'

Data Types: `char`

See Also

`SimulinkRealTime.copyFileToTarget` | `SimulinkRealTime.fileSystem.cd` | `SimulinkRealTime.fileSystem.dir` | `SimulinkRealTime.fileSystem.pwd`

Introduced in R2014a

SimulinkRealTime.copyFileToTarget

Copy file from development computer to target computer

Syntax

```
SimulinkRealTime.copyFileToTarget(file_name)  
SimulinkRealTime.copyFileToTarget(target_obj, file_name)
```

Description

`SimulinkRealTime.copyFileToTarget(file_name)` copies file `file_name` from the development computer to the default target computer.

`SimulinkRealTime.copyFileToTarget(target_obj, file_name)` copies file `file_name` from the development computer to the target computer represented by `target_obj`.

Examples

Copy File to Default Target Computer Top Folder

Copy file from current folder on development computer to top folder on default target computer.

```
SimulinkRealTime.copyFileToTarget('data.dat')
```

Copy File to Specified Target Computer by Full Path

Copy file from current folder on development computer to full path location on target computer `TargetPC1`.

```
tg = slrt('TargetPC1');
```

```
SimulinkRealTime.copyFileToTarget(tg, 'c:\xpcosc\data1.dat')
```

Input Arguments

target_obj — Name of a target computer or a variable containing a target computer object
character vector | object

If the argument is a character vector, the character vector must contain the name assigned to a previously configured target computer.

If the argument is a variable containing an object, the object must be a `SimulinkRealTime.target` object representing a previously configured target computer.

Example: 'TargetPC1'

Example: tg

Data Types: char | struct

file_name — Name of a file in the current folder on the development computer
file name character vector | full path name character vector

The file being copied must exist in the current folder on the development computer.

If the argument is a file name, the file is copied to the current folder on the target computer, as indicated by the function `SimulinkFileSystem.pwd`.

If the argument is a path name, the file portion of the path name is extracted as the development computer file name. The file is copied to the location indicated by the path name. The folder must exist on the target computer.

Example: 'myFile.txt'

Example: 'c:\subDir\myFile.txt'

Data Types: char

See Also

`SimulinkRealTime.copyFileToHost` | `SimulinkRealTime.fileSystem.cd` |
`SimulinkRealTime.fileSystem.dir` | `SimulinkRealTime.fileSystem.pwd`

Introduced in R2014a

Crash Info

Retrieve information about a target computer CPU exception

Description

Creates an object that reads a crash file from target computer

Some target computers contain hardware that can retain information in memory from before a software restart. If these computers also contain a hard drive, they can save crash data after a fatal error.

Caution After a fatal error, do not restart the computer manually by using the boot or power switch. A manual restart prevents the computer from saving the crash data.

Twenty seconds after a fatal error, the target computer restarts itself and saves the crash data on the target computer hard drive. When the computer is running again, you can call the `SimulinkRealTime.crashInfo` function from the development computer to retrieve the crash data.

Create Object

`SimulinkRealTime.crashInfo`

Properties

crashData — Structure that contains crash dump data

structure

This property is read only.

Structure with the following customer-relevant fields:

- `MATLABRelease` — Version of MATLAB
- `HasException` — 1 if the CPU had an exception, otherwise 0

- `modelName` — Name of real-time application
- `MdlExecutionTime` — Stop time of model

The remaining fields are for MathWorks internal use only.

crashLocation — Structure that contains the crash location

structure

This property is read only.

Structure with the following customer-relevant fields:

- `Found` — 1 if the crash point was found, otherwise 0
- `Message` — Message describing location, one of:
 - `Found in model code`
 - `Failed to locate crash point in model code`
 - `Crash point is outside reachable address space`
- `File` — Name of crash source file
- `Line` — Line number in source file
- `Function` — Name of function that causes crash

The remaining field is for MathWorks internal use only.

The line number comes from the value that the program instruction pointer had when the kernel exception handler caught the fatal exception. The crash can come from a previous instruction and therefore from a previous line of code.

crashTime — Structure that contains time when crash occurred

structure

Structure with the following customer-relevant fields:

- `TargetTimeAtCrash` — Time of crash, according to target computer clock
- `CurrentTargetTime` — Time of call to get crash information, according to target computer clock
- `CurrentHostTime` — Time of call to get crash information, according to development computer clock

buildDir — Folder where real-time application was built

current directory (default) | character vector

Specifies the model build folder. If the current folder is not the build folder, you can set `buildDir` to a specific value. The object uses the build folder to locate the model files.

Object Functions

<code>SimulinkRealTime.crashInfo.display</code>	Display crash information
<code>SimulinkRealTime.crashInfo.update</code>	Update crash information object

Examples

Get Crash Information After CPU Exception

Create a `crashInfo` object, get its properties, display crash information.

Wait for the target computer to restart itself and display the error message.

Error: Target computer halted with an exception and restarted automatically. To get information about the exception, call `SimulinkRealTime.crashInfo` from MATLAB.

Create a `crashInfo` object.

```
cinfo_object = SimulinkRealTime.crashInfo('TargetPC1')
```

```
Crash information object saved as C:\Users\AppData\Local\...
Temp\SLRTCashInfo_2016_28_20_56_00_33.mat
```

```
----- Crash report -----
Crash time:      28-Jun-2016 20:56:00. Current target ...
                computer time: 28-Jun-2016 20:58:00
Model:          testmodel
Crash address:  2003B643
Model base:    20030000
File:          c:\pdbparsing\test_sfuns.c, line 106
Function:      mdlOutputs
Message:      Found in model code
```

For technical support, send the `SLRTCashInfo*.mat` file to MathWorks® Support (www.mathworks.com/support).

See Also

`SimulinkRealTime.getSupportInfo`

More About

- “Find Simulink Real-Time Support”
- “Error from Crash Info Function”

External Websites

- www.mathworks.com/support

Introduced in R2016b

SimulinkRealTime.crashInfo

Create crash information object

Syntax

```
cinfo_object = SimulinkRealTime.crashInfo(target_name)
cinfo_object = SimulinkRealTime.crashInfo(target_object)
cinfo_object = SimulinkRealTime.crashInfo(settings_object)
```

Description

`cinfo_object = SimulinkRealTime.crashInfo(target_name)` creates and returns a crash information object.

If a CPU exception occurred, it calls `SimulinkRealTime.crashInfo.update` and `SimulinkRealTime.crashInfo.display` to print the crash information.

If a CPU exception did not occur, `SimulinkRealTime.crashInfo` produces an error message.

`cinfo_object = SimulinkRealTime.crashInfo(target_object)` and `cinfo_object = SimulinkRealTime.crashInfo(settings_object)` create and return a crash information object.

If a CPU exception occurred, it calls `SimulinkRealTime.crashInfo.update` and `SimulinkRealTime.crashInfo.display` to print the crash information.

If a CPU exception did not occur, `SimulinkRealTime.crashInfo` produces an error message.

Examples

Get Crash Information After CPU Exception by Target Computer Name

Create a `crashInfo` object by name and display crash information.

Wait for the target computer to restart itself and display the error message.

Error: Target computer halted with an exception and restarted automatically. To get information about the exception, call `SimulinkRealTime.crashInfo` from MATLAB.

Create a `crashInfo` object.

```
cinfo_object = SimulinkRealTime.crashInfo('TargetPC1')
```

```
Crash information object saved as C:\Users\AppData\Local\...  
Temp\SLRTCashInfo_2016_28_20_56_00_33.mat
```

```
----- Crash report -----  
Crash time:      28-Jun-2016 20:56:00. Current target ...  
  computer time: 28-Jun-2016 20:58:00  
Model:          testmodel  
Crash address:  2003B643  
Model base:     20030000  
File:           c:\pdbparsing\test_sfund.c, line 106  
Function:       mdlOutputs  
Message:        Found in model code
```

Get Crash Information After CPU Exception by Target Object

Create a `crashInfo` object by target object and display crash information.

Create and display a `crashInfo` object.

```
target_object = slrt;  
cinfo_object = SimulinkRealTime.crashInfo(target_object)
```

```
Crash information object saved as C:\Users\AppData\Local\...  
Temp\SLRTCashInfo_2016_28_20_56_00_33.mat
```

```
----- Crash report -----  
Crash time:      28-Jun-2016 20:56:00. Current target ...  
  computer time: 28-Jun-2016 20:58:00  
Model:          testmodel  
Crash address:  2003B643  
Model base:     20030000  
File:           c:\pdbparsing\test_sfund.c, line 106  
Function:       mdlOutputs  
Message:        Found in model code
```

Get Crash Information After CPU Exception by Settings Object

Create a `crashInfo` object by settings object and display crash information.

Create and display a `crashInfo` object.

```
settings_object = SimulinkRealTime.getTargetSettings('TargetPC1');
cinfo_object = SimulinkRealTime.crashInfo(settings_object)
```

```
Crash information object saved as C:\Users\AppData\Local\...
Temp\SLRTCashInfo_2016_28_20_56_00_33.mat
```

```
----- Crash report -----
Crash time:      28-Jun-2016 20:56:00. Current target ...
  computer time: 28-Jun-2016 20:58:00
Model:          testmodel
Crash address:  2003B643
Model base:     20030000
File:           c:\pdbparsing\test_sfun.c, line 106
Function:       mdlOutputs
Message:        Found in model code
```

Input Arguments

target_name — Name of target computer

character vector

Name of target computer that had a CPU exception.

Example: 'TargetPC1'

target_object — Object representing target computer that had a CPU exception

`SimulinkRealTime.target` object

Object representing the target computer.

Data Types: `struct`

settings_object — Settings object representing target computer that had a CPU exception

`SimulinkRealTime.targetSettings` object

Object containing target computer environment settings.

Data Types: `struct`

Output Arguments

cinfo_object – Object representing crash information
structure

Object that provides properties and functions for accessing crash information.

More About

- “Error from Crash Info Function”

See Also

Crash Info | `SimulinkRealTime.crashInfo.display` | `SimulinkRealTime.crashInfo.update`

Introduced in R2016b

SimulinkRealTime.crashInfo.display

Display crash information

Syntax

```
display(cinfo_object)
```

Description

`display(cinfo_object)` prints crash information in the MATLAB Command Window.

Examples

Display Crash Information

Display crash information from preexisting crash information object.

```
display(cinfo_object);
```

```
----- Crash report -----  
Crash time:      28-Jun-2016 20:56:00. Current target ...  
  computer time: 28-Jun-2016 20:58:00  
Model:          testmodel  
Crash address:  2003B643  
Model base:    20030000  
File:          c:\pdbparsing\test_sfun.c, line 106  
Function:      mdlOutputs  
Message:      Found in model code
```

Input Arguments

cinfo_object — Object representing crash information

structure

Object that provides properties and functions for accessing crash information.

More About

- “Error from Crash Info Function”

See Also

Crash Info

Introduced in R2016b

SimulinkRealTime.crashInfo.update

Update crash information object

Syntax

```
update(cinfo_object)
```

Description

`update(cinfo_object)` retrieves the results of a new CPU exception with a preexisting crash information object. It prints the location of the crash information file. To display the new crash information, call `SimulinkRealTime.crashInfo.display`.

Examples

Update and Display New Crash Information

After a new CPU exception, update and display a preexisting crash information object.

Wait for the target computer to restart itself and display the error message.

```
Error: Target computer halted with an exception and restarted
automatically. To get information about the exception, call
SimulinkRealTime.crashInfo from MATLAB.
```

Update a preexisting `crashInfo` object.

```
update(cinfo_object);
```

```
Crash information object saved as C:\Users\AppData\Local\...
Temp\SLRTCashInfo_2016_28_20_56_00_33.mat
```

Display the new crash information.

```
display(cinfo_object);
```

```
----- Crash report -----
```

```
Crash time:      28-Jun-2016 20:56:00. Current target ...  
  computer time: 28-Jun-2016 20:58:00  
Model:          testmodel  
Crash address:  2003B643  
Model base:    20030000  
File:          c:\pdbparsing\test_sfund.c, line 106  
Function:      mdlOutputs  
Message:      Found in model code
```

Input Arguments

crash_info_object — Object representing the crash information

structure

Object that provides properties and functions for accessing crash information.

More About

- “Error from Crash Info Function”

See Also

Crash Info | `SimulinkRealTime.crashInfo.display`

Introduced in R2016b

SimulinkRealTime.createBootImage

Create Simulink Real-Time boot disk or DOS Loader files

Syntax

```
SimulinkRealTime.createBootImage  
SimulinkRealTime.createBootImage(target_computer_name)  
SimulinkRealTime.createBootImage(target_settings_object)  
SimulinkRealTime.createBootImage(target_object)
```

Description

`SimulinkRealTime.createBootImage` creates a boot image for the default target computer. The form of the boot image depends upon the value of the `TargetBoot` environment property.

- **BootFloppy** — To create a boot floppy disk, the software prompts you to insert an empty formatted disk into the drive. The software writes the kernel image onto the disk and displays a summary of the creation process.
- **CDBoot** — To create a CD or DVD boot disk, the software prompts you to insert an empty formatted CD or DVD into the drive. The software writes the kernel image onto the CD or DVD and displays a summary of the creation process.
- **NetworkBoot** — To create a network boot image, the software starts the network boot server process.
- **DOSLoader** — To create DOS Loader files, the software writes kernel image and DOS Loader files into a designated location on the development computer. You can then copy the files to the target computer hard drive, to a floppy disk, or to a flash drive.
- **StandAlone** — To create files for a standalone real-time application, you must separately compile and download a combined kernel and real-time application. `SimulinkRealTime.createBootImage` does not generate a standalone application.

To update the `TargetBoot` environment property:

```
tg = SimulinkRealTime.getTargetSettings
```

```
tg.TargetBoot = new_value
```

If you update the environment, you must update the boot image with the function `SimulinkRealTime.createBootImage`.

`SimulinkRealTime.createBootImage(target_computer_name)` creates a boot image for the target computer indicated by the `target_name` character vector.

`SimulinkRealTime.createBootImage(target_settings_object)` creates a boot image for the target computer indicated by the `target_settings_object`.

`SimulinkRealTime.createBootImage(target_object)` creates a boot image for the target computer indicated by `target_object`.

Examples

Create Boot Image for Default Target Computer

Create boot image for default target computer.

```
SimulinkRealTime.createBootImage
```

Create Boot Image for Named Target Computer

Create boot image for target computer 'TargetPC1'.

```
SimulinkRealTime.createBootImage('TargetPC1')
```

Create Boot Image for Target Computer Settings Object

Create boot image for target computer represented by settings object `target_settings_object`.

```
target_settings_object = ...  
    SimulinkRealTime.getTargetSettings('TargetPC1');  
SimulinkRealTime.createBootImage(target_settings_object)
```

Create Boot Image for Target Computer Runtime Object

Create boot image for target computer represented by run-time target object `target_object`.


```
target_object = SimulinkRealTime.target('TargetPC1');  
SimulinkRealTime.createBootImage(target_object)
```

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

target_settings_object — Object representing settings for specific target computer

object variable

Object of the type returned by `SimulinkRealTime.addTarget` or `SimulinkRealTime.getTargetSettings` that represents the settings of the target computer.

Example:

Data Types: struct

target_object — Object representing target computer

`SimulinkRealTime.target` object

Object that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required Ethernet link settings.

Example: tg

Data Types: struct

More About

- “Target Boot Methods”
- “Command-Line Target Boot Methods”

See Also

`SimulinkRealTime.addTarget` | `SimulinkRealTime.getTargetSettings` |
`SimulinkRealTime.target` | `slrt` | Target Settings Properties

Introduced in R2014a

SimulinkRealTime.getSupportInfo

Diagnostic information to troubleshoot configuration issues

Syntax

```
summary = SimulinkRealTime.getSupportInfo  
summary = SimulinkRealTime.getSupportInfo(modelname)
```

Description

`summary = SimulinkRealTime.getSupportInfo` generates diagnostic information for troubleshooting Simulink Real-Time issues. The function saves the information in the file `slrtinfo.m` in the current folder. If `slrtinfo.m` exists, the function overwrites it with the new information. The function returns a structure containing key diagnostic information.

If the target computer halted with a fatal error and saved crash data on its hard drive, `SimulinkRealTime.getSupportInfo` loads the crash data into a file on the development computer and reports the path to that file.

Calling `SimulinkRealTime.getSupportInfo` unloads your model and runs the diagnostic test `slrttest`. Before calling this function, stop executing your real-time application and unload it.

`SimulinkRealTime.getSupportInfo` can record information that is sensitive to your organization. Review this information before disclosing it to MathWorks.

`summary = SimulinkRealTime.getSupportInfo(modelname)` generates and returns the same information as the function does when it is called without an argument. In addition, it generates the file `SLRTDebug.m` in the current folder. `SLRTDebug.m` contains the Simulink Configuration Parameter settings for model `modelname`.

Examples

Target Computer Information

Get diagnostic information about a functioning target computer.

```
summary = SimulinkRealTime.getSupportInfo

----- File created using the Simulink Real-Time ...
      support utility GETSUPPORTINFO -----

%% ----- General Information -----

%% Current Time & Date: 15-Jun-2016 09:50:37
.
.
.
%% SLRTTEST test:
Executing SLRTTEST
This might take a couple of minutes ...
.
.
.
### Test Suite successfully finished

%% -----End test-----

This information has been saved in the text file slrtinfo.m ...
in the current directory.
Please attach this text file to the Service Request you ...
create at:'http://www.mathworks.com/support'

Note: slrtinfo.m may contain sensitive information. Please ...
review before sending to MathWorks.

summary =

    struct with fields:

        date: '29-Jun-2016 17:11:15'
        ver: [1x88 struct]
.
.
.
        getPCIInfo: [1x17 struct]
        cpuInfo: 'System Information...'
        crashStatus: 0
        crashInfo: 0
```

This function generates the file `slrtinfo.m` in the current folder.

Target Computer and Model Information

Get diagnostic information about a functioning target computer and real-time application.

```
summary = SimulinkRealTime.getSupportInfo('testmodel')
----- File created using the Simulink Real-Time ...
        support utility GETSUPPORTINFO -----
%% ----- General Information -----
%% Current Time & Date: 15-Jun-2016 09:50:37
.
.
.
%% SLRTTEST test:
Executing SLRTTEST
This might take a couple of minutes ...
.
.
.
### Test Suite successfully finished

%% -----End test-----

This information has been saved in the text file slrtinfo.m ...
in the current directory.
Please attach this text file to the Service Request you ...
create at:'http://www.mathworks.com/support'

Note: slrtinfo.m may contain sensitive information. Please ...
review before sending to MathWorks.

summary =

    struct with fields:

        date: '29-Jun-2016 17:11:15'
        ver: [1x88 struct]
.
.
.
```

```
getPCIInfo: [1x17 struct]
  cpuInfo: 'System Information...'
crashStatus: 0
  crashInfo: 0
```

This function generates the files `slrtinfo.m` and `SLRTDebug.m` in the current folder.

Target Computer and Model Information After Fatal Error

Get diagnostic information about a functioning target computer and real-time application after a fatal error and an automatic restart.

Wait for the target computer to restart itself and display the error message.

```
Error: Target computer halted with an exception and restarted
automatically. To get information about the exception, call
SimulinkRealTime.crashInfo from MATLAB.
```

Call `getSupportInfo` to get full information about the target computer and real-time application.

```
summary = SimulinkRealTime.getSupportInfo('testmodel')
----- File created using the Simulink Real-Time ...
  support utility GETSUPPORTINFO -----
%% ----- General Information -----
%% Current Time & Date: 15-Jun-2016 09:50:37
.
.
.
%% ----- Target Crash Information: -----
1
Crash information object saved as C:\Users\AppData\Local\...
Temp\SLRTCashInfo_2016_28_20_56_00_33.mat
----- Crash report -----
Crash time:      28-Jun-2016 20:56:00. Current target ...
  computer time: 28-Jun-2016 20:58:00
Model:          testmodel
Crash address:  2003B643
Model base:     20030000
File:           c:\pdbparsing\test_sfunk.c, line 106
```

```
Function:          mdlOutputs
Message:          Found in model code
```

```
%% SLRTTEST test:
Executing SLRTTEST
This might take a couple of minutes ...
```

```
.
.
.
### Test Suite successfully finished
```

```
%% -----End test-----
```

```
This information has been saved in the text file slrtinfo.m ...
in the current directory.
Please attach this text file to the Service Request you ...
create at: 'http://www.mathworks.com/support'
```

```
Note: slrtinfo.m may contain sensitive information. Please ...
review before sending to MathWorks.
```

```
summary =
```

```
struct with fields:
```

```
date: '29-Jun-2016 17:11:15'
ver: [1x88 struct]
```

```
.
.
.
```

```
getPCIInfo: [1x17 struct]
cpuInfo: 'System Information...'
crashStatus: 1
crashInfo: [1x1 SimulinkRealTime.crashInfo]
```

This function generates the files `slrtinfo.m` and `SLRTDebug.m` in the current folder. It generates the file `SLRTCashInfo*.mat` on the development computer hard drive.

Input Arguments

modelName — Name of the model being executed

usrname

Do not include a file extension in **modelname**.

Example: 'xpcosc'

Data Types: char

Output Arguments

summary — Key diagnostic information

struct

The function returns a struct containing the following information:

- **date** — The current date
- **ver** — Names and versions of the installed MathWorks products
- **path** — The Windows path
- **matlabroot** — The location where MATLAB is installed.
- **pwd** — The current folder.
- **hostname** — The name of the development computer
- **dosversion** — The version of Windows installed on the development computer
- **antivirus** — Information about antivirus software installed on the development computer
- **slrtroot** — The location where Simulink Real-Time is installed
- **TargetSettings** — The current target computer settings
- **Compiler** — The name of the compiler installed on the development computer
- **CompilerPath** — The location of the compiler installed on the development computer
- **Kernelnames, Kernelinfo** — Internal kernel information
- **ArpEntries, Selfping, DosTargetPing, ARPEntriesAfterPing** — Kernel communication information
- **getPCIInfo** — Information about devices on the target computer PCI bus
- **cpuInfo** — Information about the target computer CPU
- **crashStatus** — 1 if the target computer had a fatal error, and otherwise 0.
- **crashInfo** — Information about the fatal error if the target computer had a fatal error, and otherwise does not appear.

More About

- “Find Simulink Real-Time Support”

See Also

Crash Info

Introduced in R2014a

SimulinkRealTime.getTargetSettings

Get target computer environment settings

Syntax

```
SimulinkRealTime.getTargetSettings
SimulinkRealTime.getTargetSettings(target_computer_name)
settings_object = SimulinkRealTime.getTargetSettings( ___ )

SimulinkRealTime.getTargetSettings('-all')
settings_object_vector = SimulinkRealTime.getTargetSettings('-all')
```

Description

`SimulinkRealTime.getTargetSettings` displays the environment settings for the default computer.

`SimulinkRealTime.getTargetSettings(target_computer_name)` displays the environment settings for a particular target computer.

`settings_object = SimulinkRealTime.getTargetSettings(___)` returns an environment object representing a target computer.

`SimulinkRealTime.getTargetSettings('-all')` displays a list of environment objects representing all defined target computers.

`settings_object_vector = SimulinkRealTime.getTargetSettings('-all')` returns a vector of environment objects representing all target computers.

Examples

Display Settings for Default Target

Display environment settings for default target computer.

```
SimulinkRealTime.getTargetSettings
```

Simulink Real-Time Target Settings

```

Name : TargetPC1

TargetRAMSizeMB : Auto
MaxModelSize : 1MB
SecondaryIDE : off
MulticoreSupport : on
LegacyMultiCoreConfig : on
USBSupport : on
ShowHardware : off
EthernetIndex : 0

TcpIpTargetAddress : 10.10.10.15
TcpIpTargetPort : 22222
TcpIpSubNetMask : 255.255.255.0
TcpIpGateway : 255.255.255.255
TcpIpTargetDriver : I8254x
TcpIpTargetBusType : PCI

TargetScope : Enabled

TargetBoot : NetworkBoot
TargetMACAddress : 00:01:29:55:3c:bb

```

Display Settings for Specific Target

Display environment settings for a specific target computer.

```
SimulinkRealTime.getTargetSettings('TargetPC2')
```

Simulink Real-Time Target Settings

```

Name : TargetPC2

TargetRAMSizeMB : Auto
MaxModelSize : 1MB
SecondaryIDE : off
MulticoreSupport : on
LegacyMultiCoreConfig : on
USBSupport : on
ShowHardware : off
EthernetIndex : 0

TcpIpTargetAddress : 10.10.10.30

```

```
TcpIpTargetPort      : 22222
TcpIpSubNetMask     : 255.255.255.0
TcpIpGateway        : 255.255.255.255
TcpIpTargetDriver   : I8254x
TcpIpTargetBusType  : PCI

TargetScope         : Enabled

TargetBoot          : NetworkBoot
TargetMACAddress    : 90:e2:ba:17:5d:15
```

Display Settings for All Targets

Display environment settings for all target computers.

```
SimulinkRealTime.getTargetSettings('-all')
```

```
NumTargets: 2
Targets   : Name           Communication Settings . . .
           TargetPC1 (Default) TcpIp:10.10.10.15:22222 . . .
           TargetPC2      TcpIp:10.10.10.30:22222 . . .
```

```
Simulink Real-Time Target Settings
```

```
      Name           : TargetPC1
      .
      .
      .
      TcpIpTargetAddress : 10.10.10.15
      .
      .
      .
      TargetBoot       : NetworkBoot
      TargetMACAddress : 00:01:29:55:3c:bb
```

```
Simulink Real-Time Target Settings
```

```
      Name           : TargetPC2
      .
      .
      .
```

```

    TcpIpTargetAddress      : 10.10.10.30
    .
    .
    .
    TargetBoot              : NetworkBoot
    TargetMACAddress        : 90:e2:ba:17:5d:15

```

Access Settings for Specific Target

Retrieve an environment settings object for a specific target computer. Use it to access a setting.

```

settings_object = SimulinkRealTime.getTargetSettings('TargetPC1');
settings_object.TcpIpTargetAddress

```

```

ans =

10.10.10.15

```

Access Settings for Multiple Targets

Loop through vector of environment settings objects. Print name and communication settings.

```

sov = SimulinkRealTime.getTargetSettings('-all');
ii = 1;
while ii <= length(sov)
    disp(sprintf('%s TcpIpTargetAddress is %s', ...
                sov(ii).Name, sov(ii).TcpIpTargetAddress))
    ii = ii + 1;
end

```

```

TargetPC1 TcpIpTargetAddress is 10.10.10.15
TargetPC2 TcpIpTargetAddress is 10.10.10.30

```

Input Arguments

target_computer_name — Name of target computer
character vector

The name-character vector of a target computer.

Example: 'TargetPC1'

Data Types: char

Output Arguments

settings_object — Settings object representing target computer

SimulinkRealTime.targetSettings object

Object containing target computer environment settings.

Data Types: struct

settings_object_vector — Vector of settings objects representing target computers

vector

Vector of objects containing target computer environment settings representing one or more target computers

Data Types: struct

See Also

Target Settings Properties

Introduced in R2014a

SimulinkRealTime.pingTarget

Test communication between development and target computers

Syntax

```
SimulinkRealTime.pingTarget
```

```
SimulinkRealTime.pingTarget(target_computer_name)
```

Description

`SimulinkRealTime.pingTarget` without an argument returns **success** if the development computer and the default target computer can communicate using the settings for the default computer. Otherwise, returns **failed**.

`SimulinkRealTime.pingTarget(target_computer_name)` returns **success** if the development computer can communicate with target computer `target_computer_name` using the settings for target computer `target_computer_name`. Otherwise, returns **failed**.

Enclose the argument in single quotes ('TargetPC1').

Examples

Check communication with default target computer

```
SimulinkRealTime.pingTarget
```

Check communication with specified target computer

```
SimulinkRealTime.pingTarget('TargetPC1')
```

Input Arguments

target_computer_name — Name of specific target computer
'TargetPC1' | 'TargetPC2' | ...

Name property of a particular target computer environment object. The default name is 'TargetPC1'.

Example: `TargetPC1`

Data Types: `char`

Introduced in R2014a

SimulinkRealTime.removeTarget

Remove environment data associated with target name

Syntax

```
SimulinkRealTime.removeTarget('target_name')
```

Description

`SimulinkRealTime.removeTarget('target_name')` removes the definitions and settings for the target computer represented by `'target_name'` from the system. The target objects associated with that target become invalid. If you remove the environment data for the default target computer, the next target object becomes the default target computer. Do not remove the environment data for the last target computer.

Examples

Remove the environment data for 'TargetPC2' from the system:

```
SimulinkRealTime.removeTarget('TargetPC2')
```

See Also

`SimulinkRealTime.addTarget` | `SimulinkRealTime.getTargetSettings`

Introduced in R2014a

SimulinkRealTime.utils.bytes2file

Generate file for use by real-time From File block

Syntax

```
SimulinkRealTime.utils.bytes2file(filename, var1, . . . , varX)
```

Description

`SimulinkRealTime.utils.bytes2file(filename, var1, . . . , varX)` generates a file for use by the real-time From File block. The From File block outputs one column of variables `var1`, . . . , `varX` from file `filename` at every time step.

Variables `var1`, . . . , `varX` must be matrices in column-major format and have the same number of columns. The number of rows and the data types of the matrix elements can be different.

If the data is organized in row-major format (a row, not a column, refers to a time step), transpose the variable and pass the transpose to `SimulinkRealTime.utils.bytes2file`. To optimize file writes, organize the data in columns.

Examples

Errorval and Velocity in Column-Major Format

From File outputs two variables `errorval` and `velocity` at every time step from 1 to N. Each variable is in column-major format.

Variable `errorval` has class `'single'` and dimensions `[1 x N]`. Variable `velocity` has class `'double'` and dimensions `[3 x N]`.

```
SimulinkRealTime.utils.bytes2file('myfile', errorval, velocity)
```

Set up the real-time From File block to output 28 bytes at every sample time ((1 * sizeof('single') + 3 * sizeof('double'))).

Errorval and Velocity in Row-Major Format

From File outputs two variables `errorval` and `velocity` at every time step from 1 to N. Each variable is in row-major format.

Variable `errorval` has class 'single' and dimensions [N x 1]. Variable `velocity` has class 'double' and dimensions [N x 3].

```
SimulinkRealTime.utils.bytes2file('myfile', ...
                                transpose(errorval), ...
                                transpose(velocity));
```

Set up the real-time From File block to output 28 bytes at every sample time ((1 * sizeof('single') + 3 * sizeof('double'))).

Input Arguments

filename — Name of the data file

character vector

The data file contains columns of data to be output to the model.

Example: 'myfile'

Data Types: char

var1, . . . , varX — X arguments, each in column-major format

real and integer

The X arguments each provide columns of data to be output to the model.

Example: `errorval`, `velocity`

Data Types: single | double | int8 | int16 | int32 | uint8 | uint16 | uint32

See Also

From File

Introduced in R2014a

SimulinkRealTime.utils.createInstrumentationModel

Construct skeleton for user interface model

Syntax

```
SimulinkRealTime.utils.createInstrumentationModel(system_name)
```

Description

`SimulinkRealTime.utils.createInstrumentationModel(system_name)` generates a skeleton Simulink instrumentation model containing To Target and From Target blocks. The model is based on tagged block parameters and tagged signals defined in the Simulink Real-Time model used to build the real-time application.

Examples

Generate an interface model

```
SimulinkRealTime.utils.createInstrumentationModel('xpcosc')
```

Input Arguments

system_name — Name of system for which to create an interface model
'xpcosc'

Model must contain tagged signals or block parameters.

Data Types: char

Introduced in R2014a

SimulinkRealTime.utils.getFileScopeData

Read real-time Scope file format data

Syntax

```
matlab_data = SimulinkRealTime.utils.getFileScopeData(slrtfile_name)
matlab_data = SimulinkRealTime.utils.getFileScopeData(slrtfile_data)
```

Description

`matlab_data = SimulinkRealTime.utils.getFileScopeData(slrtfile_name)` takes as an argument the name of a development computer file containing a vector of byte data (`uint8`). Before using this function, copy the file from the target computer using the `SimulinkRealTime.copyFileToHost` method.

`matlab_data = SimulinkRealTime.utils.getFileScopeData(slrtfile_data)` takes as an argument a MATLAB variable containing a vector of byte data (`uint8`). Before using this function, load the data into memory from a file on the target file system using the `SimulinkRealTime.fileSystem.fread` method.

Examples

Using `slrtfile_name` argument to read file and plot results

Upload file 'data.dat' to the host. Read the file on the host. Plot the results.

Upload file 'data.dat' from the target computer to the development computer.

```
SimulinkRealTime.copyFileToHost('data.dat')
```

Read the file and process its data into MATLAB format.

```
matlab_data = SimulinkRealTime.utils.getFileScopeData('data.dat');
```

Plot the signal data (column 1) on the Y axis against time (column 2) on the X axis.

```
plot(matlab_data.data(:,2), matlab_data.data(:,1))  
xlabel(matlab_data.signalNames(2))  
ylabel(matlab_data.signalNames(1))
```

Using `slrtfile_data` argument to store data, convert data to MATLAB format, and plot results

Read file 'data.dat' on the target computer from the host. Store the data in a MATLAB workspace variable. Convert the data to MATLAB format. Plot the results.

Read file 'data.dat' from the development computer using file system commands.

```
fs = SimulinkRealTime.fileSystem;  
h = fopen(fs, 'data.dat');  
slrtfile_data = fread(fs, h);  
fclose(fs,h)
```

Process data from the workspace variable into MATLAB format.

```
matlab_data =  
    SimulinkRealTime.utils.getFileScopeData(slrtfile_data);
```

Plot the signal data (column 1) on the Y axis against time (column 2) on the X axis.

```
plot(matlab_data.data(:,2), matlab_data.data(:,1))  
xlabel(matlab_data.signalNames(2))  
ylabel(matlab_data.signalNames(1))
```

Input Arguments

slrtfile_name — Name of file from which to read real-time Scope file format data
'data.dat'

File must contain a vector of `uint8` data.

Data Types: `char`

slrtfile_data — Workspace variable containing real-time Scope file format data
vector

Data Types: `uint8`

Output Arguments

matlab_data — State and time data for plotting

structure

The state and time data is stored in a structure containing six fields. The key fields are `numSignals`, `data`, and `signalNames`.

version — Version code

0 (default) | double

Internal

sector — Sector of data file

0 (default) | double

Internal

headersize — Number of bytes of data file header

512 (default) | double

Internal

numSignals — Number of columns containing signal and time data

double

If N signals are connected to the real-time Scope block, `numSignals` = $N + 1$.

data — Columns containing signal and time data

double array

The `data` array contains `numSignals` columns. The first N columns represent signal state data. The last column contains the time at which the state data is captured.

The `data` array contains as many rows as there are data points.

signalNames — Names of columns containing signal and time data

cell vector

The `signalNames` vector contains `numSignals` elements. The first N elements are signal names. The last element is the character vector `Time`.

See Also

File System | Scope | `SimulinkRealTime.copyFileToHost`

Introduced in R2014a

SimulinkRealTime.utils.getTargetSystemTime

Gets the current value of the target computer system clock

Syntax

```
date_vector = SimulinkRealTime.utils.getTargetSystemTime
date_vector = SimulinkRealTime.utils.getTargetSystemTime(
target_object)
```

Description

`date_vector = SimulinkRealTime.utils.getTargetSystemTime` returns the system time of the default target computer as a date vector. The target computer must be running and in communication with the development computer.

`date_vector = SimulinkRealTime.utils.getTargetSystemTime(target_object)` returns the system time of the specified target computer as a date vector.

Examples

Get System Time of Default Target Computer

Return the system time of the default target computer as a date vector.

```
date_vector = SimulinkRealTime.utils.getTargetSystemTime
```

```
date_vector =
```

```
Columns 1 through 4
```

```
    2015         11         4         14
```

```
Columns 5 through 6
```

37

34

Get System Time of Specified Target Computer

Return the system time of target computer 'TargetPC1' as a date vector.

```
target_object = SimulinkRealTime.target('TargetPC1');  
date_vector = ...  
    SimulinkRealTime.utils.getTargetSystemTime(target_object)
```

```
date_vector =
```

```
Columns 1 through 4
```

```
    2015         11         4         14
```

```
Columns 5 through 6
```

```
    39         45
```

Input Arguments

target_object — Object representing target computer

SimulinkRealTime.target object

Object that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required Ethernet link settings.

Example: tg

Data Types: struct

Output Arguments

date_vector — Date and time vector

datevec

Date and time as returned by the datevec function

Example: [2015, 11, 5, 14, 15, 0]

Data Types: double

See Also

SimulinkRealTime.utils.setTargetSystemTime

Introduced in R2016a

SimulinkRealTime.utils.minimumSampleTime

Determine the minimum sample time at which a model can run

Syntax

```
minTs = SimulinkRealTime.utils.minimumSampleTime(model_name)
minTs = SimulinkRealTime.utils.minimumSampleTime(model_name, '-
cleanup')
```

Description

`minTs = SimulinkRealTime.utils.minimumSampleTime(model_name)` executes the model in real time on a target computer and returns the minimum sample time at which it can run.

The target computer must be running and connected to the development computer. The function builds the model and downloads it automatically to the target computer.

`minTs = SimulinkRealTime.utils.minimumSampleTime(model_name, '-cleanup')` executes the model in real time on a target computer and returns the minimum sample time at which it can run.

The target computer must be running and connected to the development computer. The function builds the model and downloads it automatically to the target computer. When execution is complete, the function deletes the build files.

Examples

Determine Minimum Sample Time

Determines the minimum sample time of model `xpcosc`.

```
minTs = SimulinkRealTime.utils.minimumSampleTime('xpcosc')
minTs =
```

```
8.4727e-06
```

To avoid CPU overruns, set your model sample time to a value slightly above the lower limit, for example to $10e-6$.

Determine Minimum Sample Time and Delete Build Files

Determines the minimum sample time of model `xpcosc`, and then cleans up the build folder.

```
minTs = SimulinkRealTime.utils.minimumSampleTime('xpcosc', ...  
        '-cleanup')
```

```
minTs =
```

```
8.4727e-06
```

To avoid CPU overruns, set your model sample time to a value slightly above the lower limit, for example to $10e-6$.

Input Arguments

model_name — Name of the model

character vector

Enclose the model name character vector in single quotation marks.

Example: 'xpcosc'

Data Types: char

Output Arguments

minTs — Minimum sample time

double

The minimum sample time at which the function executed the model. To avoid the overloads that random variations can cause, set your model sample time to a value slightly above the minimum sample time.

More About

- “Profiling and Optimization”
- “Improve Performance of Multirate Model”

Introduced in R2016a

SimulinkRealTime.utils.setTargetSystemTime

Sets the value of the target computer system clock

Syntax

```
SimulinkRealTime.utils.setTargetSystemTime
SimulinkRealTime.utils.setTargetSystemTime(date_vector)
SimulinkRealTime.utils.setTargetSystemTime(target_object, ___)
```

Description

`SimulinkRealTime.utils.setTargetSystemTime` sets the default target computer system time to the current value of the development computer system time (UTC). The target computer must be running and in communication with the development computer. You do not have to use the target computer keyboard or restart the target computer.

`SimulinkRealTime.utils.setTargetSystemTime(date_vector)` sets the default target computer system time to the specified value, passed as a date vector.

`SimulinkRealTime.utils.setTargetSystemTime(target_object, ___)` sets the specified target computer system time to the specified value, passed as a date vector.

Examples

Set Default Target Computer System Time to Development Computer System Time

Change system time of default target computer to the development computer system time

Show original system time.

```
date_vector = SimulinkRealTime.utils.getTargetSystemTime
```

```
date_vector =
```

Columns 1 through 4

```
        2015        11         4         19
Columns 5 through 6
        15         56
```

Change system time.

```
SimulinkRealTime.utils.setTargetSystemTime;
```

Show new system time.

```
date_vector = SimulinkRealTime.utils.getTargetSystemTime
```

```
date_vector =
```

```
Columns 1 through 4
        2015        11         4         19
Columns 5 through 6
        15         57
```

Set Default Target Computer System Time to Specified System Time

Change system time of default target computer to the specified system time

Show original system time.

```
date_vector = SimulinkRealTime.utils.getTargetSystemTime
```

```
date_vector =
```

```
Columns 1 through 4
        2015        11         4         19
Columns 5 through 6
        15         57
```

Change system time to

```
new_date_vector = [2015, 11, 5, 14, 15, 0];
```



```
SimulinkRealTime.utils.setTargetSystemTime(new_date_vector);
```

Show new system time.

```
date_vector = SimulinkRealTime.utils.getTargetSystemTime
```

```
date_vector =
```

```
Columns 1 through 4
```

```
2015      11      5      14
```

```
Columns 5 through 6
```

```
15      0
```

Set Specified Target Computer System Time to Development Computer System Time

Change system time of target computer 'TargetPC1' to the development computer system time

Show original system time.

```
target_object = SimulinkRealTime.target('TargetPC1');
```

```
date_vector = ...
```

```
    SimulinkRealTime.utils.getTargetSystemTime(target_object)
```

```
date_vector =
```

```
Columns 1 through 4
```

```
2015      11      5      14
```

```
Columns 5 through 6
```

```
15      0
```

Change system time.

```
SimulinkRealTime.utils.setTargetSystemTime(target_object);
```

Show new system time.

```
date_vector = ...
```

```
    SimulinkRealTime.utils.getTargetSystemTime(target_object)
```

```
date_vector =  
  
Columns 1 through 4  
    2015         11         4         19  
  
Columns 5 through 6  
    15         57
```

Input Arguments

target_object — Object representing target computer

SimulinkRealTime.target object

Object that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required Ethernet link settings.

Example: tg

Data Types: struct

date_vector — Date and time vector

datevec

Date and time as returned by the datevec function

Example: [2015, 11, 5, 14, 15, 0]

Data Types: double

See Also

SimulinkRealTime.utils.getTargetSystemTime

Introduced in R2016a

Target Settings Properties

Store settings related to target computer

Description

This object defines the settings for the target computer.

The settings define the communication link between the development and target computers and the properties of the target boot image created during the setup process.

To create a target computer settings object that is set to default values, use the syntax `target_object = SimulinkRealTime.addTarget(target_name)`.

```
target_object = SimulinkRealTime.addTarget('TargetPC3')
```

Simulink Real-Time Target Settings

```
Name : TargetPC3

TargetRAMSizeMB : Auto
MaxModelSize : 1MB
SecondaryIDE : off
NonPentiumSupport : off
MulticoreSupport : on
LegacyMultiCoreConfig : on
USBSupport : on
ShowHardware : off
EthernetIndex : 0

TcpIpTargetAddress :
TcpIpTargetPort : 22222
TcpIpSubNetMask : 255.255.255.0
TcpIpGateway : 255.255.255.255
TcpIpTargetDriver : Auto
TcpIpTargetBusType : PCI
TcpIpTargetISAMemPort : 0x300
TcpIpTargetISAIRQ : 5

TargetScope : Enabled

TargetBoot : BootFloppy
```

```
BootFloppyLocation      :
```

The default settings are incomplete. At a minimum, you must assign a value to `TcpIpTargetAddress`. To change this setting by assignment, use the syntax `target_object.property_name = value`.

```
target_object = SimulinkRealTime.getTargetSettings('TargetPC3');  
target_object.TcpIpTargetAddress = '10.10.10.15';
```

To read an existing setting, use the syntax `value = target_object.property_name`.

```
target_object = SimulinkRealTime.getTargetSettings('TargetPC3');  
value = target_object.TcpIpTargetAddress
```


```
value =
```

```
10.10.10.15
```

To mark a target computer as the default computer, use the syntax `setAsDefaultTarget(target_object)`.

```
target_object = SimulinkRealTime.getTargetSettings('TargetPC3');  
setAsDefaultTarget(target_object)
```

To access the target computer settings in Simulink Real-Time Explorer:

- 1 In the **Targets** pane, expand a target computer node.
- 2 In the toolbar, click the **Target Properties** button .
- 3 Expand the sections **Host-to-Target communication**, **Target settings**, or **Boot configuration**.

Host-to-Target Communication

TcpIpGateway — IP address for gateway to Ethernet link

'255.255.255.255' (default) | 'xxx.xxx.xxx.xxx'

If your development and target computers connect through a LAN that uses a gateway, you must enter a value for this property.

The default value, `255.255.255.255`, means that a gateway is not used to connect to the target computer. If your LAN does not use gateways, you do not need to change this property. Consult your system administrator for this value.

In the Simulink Real-Time Explorer **Gateway** box, type the IP address for your gateway.

Example: `env_object.TcpIpGateway = '192.168.1.1'`

TcpIpSubNetMask — Subnet mask for gateway to Ethernet link

'xxx.xxx.xxx.xxx'

In the Simulink Real-Time Explorer **Subnet mask** box, type the subnet mask of your LAN. Consult your system administrator for this value.

Example: `env_object.TcpIpSubNetMask = '255.255.255.0'`

TcpIpTargetAddress — IP address for target computer

'xxx.xxx.xxx.xxx'

In the Simulink Real-Time Explorer **IP address** box, type a valid IP address for your target computer. Consult your system administrator for this value.

Example: `env_object.TcpIpTargetAddress = '192.168.1.10'`

TcpIpTargetBusType — Bus type for Ethernet card on target computer

'PCI' (default) | 'ISA' | 'USB'

This property determines the bus type of your target computer. You do not need to define a bus type for your development computer.

Note: In R2017a, support for using ISA bus Ethernet cards to communicate between the development and target computers will cease to function. Use PCI bus or USB bus Ethernet cards instead.

If `TcpIpTargetBusType` is set to `PCI`, then the properties `TcpIpISAMemPort` and `TcpIpISAIRQ` are not used for TCP/IP communication.

If you are using an ISA bus card, set `TcpIpTargetBusType` to `ISA` and enter values for `TcpIpISAMemPort` and `TcpIpISAIRQ`.

In the Simulink Real-Time Explorer **Bus type** list, select one of `PCI` or `USB`.

Example: `env_object.TcpIpTargetBusType = 'USB'`

TcpIpTargetDriver — Driver for Ethernet card on target computer

'Auto' (default) | '3C90x' | 'I8254x' | 'I82559' | 'NE2000' | 'NS83815'
| 'R8139' | 'R8168' | 'Rhine' | 'RTLANCE' | 'SMC91C9X' | 'USBAX772' |
'USBAX172'

Use the default value ('Auto') if the target computer contains only one supported Ethernet card.

Use 'USBAX772' or 'USBAX172' if you are using bus type 'USB'.

In the Simulink Real-Time Explorer **Target driver** list, select one of THREECOM_3C90x, INTEL_I8254x, INTEL_I82559, NE2000, NS83815, R8139, R8168, Rhine, RTLANCE, SMC91C9X, USBAX772, USBAX172, or Auto.

Example: `env_object.TcpIpTargetDriver = 'USBAX172'`

TcpIpTargetISAIRQ — IRQ for Ethernet card on ISA bus target computer

'5' (default) | 'N' | '15'

IRQ values run from '5' to '15', inclusive.

If you are using an ISA bus Ethernet card, you must enter a value for `TcpIpISAIRQ`. The value must correspond to the jumper or ROM settings on the ISA bus Ethernet card.

On your ISA bus card, assign an IRQ by moving the jumpers on the card. Set the IRQ to 5, 10, or 11. If one of these settings leads to a conflict in your target computer, choose another IRQ and make the corresponding changes to your jumper settings.

Example: `env_object.TcpIpTargetISAIRQ = '11'`

TcpIpTargetISAMemPort — IRQ base address for Ethernet card on ISA bus target computer

0xNNNN

If you are using an ISA bus Ethernet card, you must enter a value for the property `TcpIpISAMemPort`. The value of this property must correspond to the jumper or ROM settings on your ISA bus Ethernet card.

On your ISA bus card, assign an 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 a conflict in your target computer results, choose another I/O port base address and make the corresponding changes to your jumper settings.

Example: `env_object.TcpIpTargetISAMemPort = '0x400'`

TcpIpTargetPort — Ethernet port on target computer

'22222'. (default) | 'xxxxx'

Typically, you do not change this value from the default. Do so only if you are using the default port ('22222') for other purposes.

Use an Ethernet port greater than '20000'. Values in this range are higher than the reserved area (telnet, ftp, ...).

```
Example: env_object.TcpIpTargetPort = '24000'
```

Target settings

EthernetIndex — Zero-based index number of Ethernet card on target computer

'0' (default) | 'n'

Unique number identifying an Ethernet card on the target computer. If the target computer has multiple Ethernet cards, you must select one of the cards for the Ethernet link. This option returns the index number of the card selected on the target computer upon starting.

```
Example: env_object.EthernetIndex = '2'
```

LegacyMultiCoreConfig — Use existing multiprocessor floating pointer structure (MPFPS) in the BIOS

'on' (default) | 'off'

When this value is 'on', the kernel uses the existing multiprocessor floating pointer structure (MPFPS) in the BIOS. When this value is 'off', the kernel uses the Advanced Configuration and Power Interface (ACPI) to query the hardware boards. The kernel uses that information to construct an MPFPS structure.

Set this value to 'off' only if your multicore target computer is fully compliant with the ACPI standard.

```
Example: env_object.LegacyMultiCoreConfig = 'off'
```

MaxModelSize — Maximum expected size of real-time application

'1MB' (default) | '4MB' | '16MB'

The maximum model size reserves the specified amount of memory on the target computer for the real-time application. Memory not used by the real-time application is used by the kernel and by the heap for data logging.

Selecting too high a value leaves less memory for data logging. Selecting too low a value does not reserve enough memory for the real-time application and creates an error. You can approximate the size of the real-time application by the size of the DLM file produced by the build process.

In the Simulink Real-Time Explorer **Model size** list, select one of 1 MB, 4 MB, or 16 MB.

Setting **Model size** is enabled for **Boot mode Stand Alone** only.

Example: `env_object.MaxModelSize = '4MB'`

MulticoreSupport — Enable use of multicore processors

'on' (default) | 'off'

Use multicore support only for a multicore target computer.

In the Simulink Real-Time Explorer, leave the **Multicore CPU** check box selected to take advantage of these processors for background tasks. Otherwise, clear it.

Example: `env_object.MulticoreSupport = 'off'`

Name — Target computer name character vector

'TargetPCN' (default) | character vector

When you create a target settings object, the software assigns it a name of the form 'TargetPCN+1'. 'TargetPCN' is the previously assigned name. You can assign a new name from the Command Window.

To rename the target computer in Simulink Real-Time Explorer, right-click the target computer node in the **MATLAB Session** tree, click **Rename**, and type the new name in the **Target environment name** box.

Example: `env_object.Name = 'NewTarget'`

NonPentiumSupport — Target computer contains legacy processor

'off' (default) | 'on'

Set only if your target computer has a 386 or 486 compatible processor. If your target computer has a Pentium or higher compatible processor, selecting this check box slows the performance of your target computer.

Note: In R2017a, the `NonPentiumSupport` property will cease to function. Use a target computer with an Intel® Pentium or AMD® K5/K6/Athlon processor.

Example: `env_object.NonPentiumSupport = 'on'`

SecondaryIDE — Enable secondary IDE disk controller

'off' (default) | 'on'

Set only if you want to use disks connected to a secondary IDE controller.

To set this parameter in Simulink Real-Time Explorer, select the **Secondary IDE** check box. Otherwise, clear it.

Example: `env_object.SecondaryIDE = 'on'`

ShowHardware — Display Ethernet card information for target computer

'off' (default) | 'on'

To display the index, bus, slot, function, and target driver for each Ethernet card on the target monitor, start the target computer with **ShowHardware** set to 'on'.

With **ShowHardware** set, after the kernel starts, the development computer cannot communicate with the target computer. When you are done gathering the information that the kernel displays, to resume normal functionality, set this property to 'off', recreate the boot image, and restart the target computer.

Example: `env_object.ShowHardware = 'on'`

TargetRAMSizeMB — Megabytes of RAM installed in target computer

'Auto' (default) | 'xxx'

Specifies the total amount of RAM, in megabytes, installed in the target computer. Target computer RAM is used for the kernel, real-time application, data logging, and other functions that use the heap.

If this property is set to 'Auto', the real-time application reads the target computer BIOS and determines the amount of memory installed in the target computer.

To allow the real-time application to determine the amount of memory in Simulink Real-Time Explorer, click **RAM size Auto**. If the real-time application cannot read the BIOS, click **Manual** and type into the **Size(MB)** box the amount of RAM, in megabytes, installed in the target computer.

Target computer memory for the real-time application executable, the kernel, and other uses is limited to a maximum of 3 GB.

Example: `env_object.ShowHardware = '2000'`

TargetScope — Display scope information graphically

'Enabled' (default) | 'Disabled'

When this property is set to 'Enabled', the target computer shows a graphical window display. When set to 'Disabled', the target computer shows a text-based view.

When the graphical display is present, you can use target scopes to view signal data graphically on the target display. You cannot do this when the text-based view is present.

Using Simulink Real-Time Explorer, to display scope information graphically, set the **Graphics mode** check box.

To display scope information as text, clear the **Graphics mode** check box.

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

Example: `env_object.TargetScope = 'Disabled'`

USBSupport — Enable USB port on target computer

'on' (default) | 'off'

Set this property to use a USB port on the target computer, for example to connect a USB mouse.

In Simulink Real-Time Explorer, to enable a USB port, select the **USB Support** check box. Otherwise, clear it.

Example: `env_object.USBSupport = 'off'`

Boot configuration**BootFloppyLocation — Drive name for creation of target boot disk**

character vector

To create a removable boot disk when the system default drive does not work, set this property.

Example: `env_object.BootFloppyLocation='D:\'`

DOSLoaderLocation — Location of DOS Loader files to start target computers from devices other than floppy disk or CD

character vector

Set this property in DOS Loader mode if the default location does not work.

Example: `env_object.DOSLoaderLocation='D:\Dosloader'`

TargetBoot — Mode of restarting target computer

'BootFloppy' (default) | 'CDBoot' | 'DOSLoader' | 'NetworkBoot' | 'StandAlone'

After making the required target settings, to create a bootable image, type `SimulinkRealTime.createTargetImage`.

In Simulink Real-Time Explorer, to create a bootable image for the specified boot mode, click **Create boot disk**.

Example: `env_object.TargetBoot='NetworkBoot'`

TargetMACAddress — Target computer MAC address for network restart

'xx:xx:xx:xx:xx:xx'

Physical target computer MAC address from which to accept start requests when starting within a dedicated network.

To update the MAC address in Simulink Real-Time Explorer, first click the **Reset** button in the **Target Properties** pane. You can then click the **Specify new MAC address** button to enter a MAC address manually in the **MAC address** box. If you do not enter a MAC address manually, the software obtains the MAC address the next time you restart the target computer.

Example: `env_object.TargetMACAddress='90:e2:ba:17:5d:15'`

See Also

`SimulinkRealTime.addTarget` | `SimulinkRealTime.getTargetSettings` | `SimulinkRealTime.targetSettings.setAsDefaultTarget`

More About

- “PCI Bus Ethernet Setup”
- “USB-to-Ethernet Setup”
- “Target Computer Settings”
- “Target Boot Methods”

Introduced in R2014a

SimulinkRealTime.targetSettings.setAsDefaultTarget

Set specific target computer environment object as default

Syntax

```
setAsDefaultTarget(settings_object)
```

Description

`setAsDefaultTarget(settings_object)` sets the specified target computer as the default target computer from the `SimulinkRealTime.targetSettings` class.

Examples

Set target computer 'TargetPC1' as the default target computer:

```
settings_object = SimulinkRealTime.getTargetSettings('TargetPC1');  
setAsDefaultTarget(settings_object)
```

Introduced in R2014a

File System

Manage folders and files on target computer

Description

The `SimulinkRealTime.fileSystem` object provides access to folders and files on the target computer.

The following limitations hold:

- You can have a maximum of eight files open on the target computer at the same time.
- The largest single file that you can create is 4 GB.
- A fully qualified folder name can have a maximum of 248 characters, including the drive letter, colon, and backslash.
- A fully qualified file name can have a maximum of 260 characters. The file part can have 12 characters—a maximum of eight characters for the file name, one character for the period, and a maximum of three characters for the file extension. If the file name is longer than eight characters, the software truncates it to six characters and adds '~1' to the end of the file name.

Create Object

`SimulinkRealTime.fileSystem`

Object Functions

`SimulinkRealTime.fileSystem.cd`
`SimulinkRealTime.fileSystem.dir`

Change folder on target computer
 List contents of current folder on target computer

`SimulinkRealTime.fileSystem.diskinfo`
`SimulinkRealTime.fileSystem.diskspace`

Target computer drive information
 Return the free space and total space on the drive, in bytes

`SimulinkRealTime.fileSystem.fclose`
`SimulinkRealTime.fileSystem.fileinfo`

Close target computer file
 Target computer file information

<code>SimulinkRealTime.fileSystem.filetable</code>	Information about open files in target computer file system
<code>SimulinkRealTime.fileSystem.fopen</code>	Open target computer file for reading and writing
<code>SimulinkRealTime.fileSystem.fread</code>	Read open target computer file
<code>SimulinkRealTime.fileSystem.fwrite</code>	Write binary data to open target computer file
<code>SimulinkRealTime.fileSystem.getfilesize</code>	Size of file on target computer
<code>SimulinkRealTime.fileSystem.mkdir</code>	Create folder on target computer
<code>SimulinkRealTime.fileSystem.pwd</code>	Path to currently active folder on target computer
<code>SimulinkRealTime.fileSystem.removefile</code>	Remove file from target computer
<code>SimulinkRealTime.fileSystem.rename</code>	Rename a file or folder in the target computer disk drive
<code>SimulinkRealTime.fileSystem.rmdir</code>	Remove empty folder from target computer
<code>SimulinkRealTime.fileSystem.selectdrive</code>	Select target computer drive

Examples

List Current Folder Contents on Default Target Computer

Create a file system object for the default target computer and use it to list the contents of the current folder

```
fsys = SimulinkRealTime.fileSystem;  
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	

Introduced in R2014a

SimulinkRealTime.fileSystem

Create file system object

Syntax

```
fileSYS_object = SimulinkRealTime.fileSystem  
fileSYS_object = SimulinkRealTime.fileSystem(target_object)
```

Description

`fileSYS_object = SimulinkRealTime.fileSystem` constructs and returns the file system object corresponding to the default target computer. If you have one target computer or if you designate a target computer as the default target computer in your system, use this form.

`fileSYS_object = SimulinkRealTime.fileSystem(target_object)` constructs and returns the file system object corresponding to the target computer that is accessible by `target_object`.

Examples

Create File System Object for Default Target Computer

Creates a file system object for the default target computer, assumed to be `TargetPC1`, and returns the disk space.

```
fsys = SimulinkRealTime.fileSystem;  
diskSpace(fsys, 'C:\')
```

```
ans =
```

```
    freeDiskSpacebytes: 5.9889e+10  
    totalDiskSpacebytes: 6.0005e+10
```

Create File System Object for Named Target Computer

Creates a file system object for target computer `TargetPC1` and returns the disk space.


```
tg = SimulinkRealTime.target('TargetPC1');
fsys = SimulinkRealTime.fileSystem(tg);
diskspace(fsys, 'C:\')

ans =

    freeDiskSpacebytes: 5.9889e+10
    totalDiskSpacebytes: 6.0005e+10
```

Input Arguments

target_object — Object representing target computer

SimulinkRealTime.target object

Object that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required Ethernet link settings.

Example: tg

Data Types: struct

Output Arguments

filesys_object — Object representing the target computer file system

SimulinkRealTime.fileSystem object

File system object created by using the SimulinkRealTime.fileSystem creation function.

The file system object represents the target computer file system. You work with the target computer file system from the development computer by using file system methods.

Example: fsys

Data Types: struct

See Also

File System

Introduced in R2014a

SimulinkRealTime.fileSystem.cd

Change folder on target computer

Syntax

```
cd(filesys_object, destination_folder)
```

Description

`cd(filesys_object, destination_folder)` changes the currently active folder on the target computer. Prints an error if the destination folder does not exist.

Examples

Change Current Folder

Using the file system object `fsys`, change the folder from the current one to one named 'logs'.

```
tg = slrt;  
fsys = SimulinkRealTime.fileSystem(tg);  
cd(fsys, 'logs')
```

Input Arguments

filesys_object – Object representing the target computer file system

`SimulinkRealTime.fileSystem` object

File system object created by using the `SimulinkRealTime.fileSystem` creation function.

The file system object represents the target computer file system. You work with the target computer file system from the development computer by using file system methods.

Example: `fsys`

Data Types: `struct`

destination_folder — Name of the destination target computer folder

character vector

Name of the target computer folder to make the current folder

Example: `'new_folder'`

Data Types: `char`

See Also

`cd` | `SimulinkRealTime.fileSystem.mkdir` |
`SimulinkRealTime.fileSystem.pwd` | `slrt`

Introduced in R2014a

SimulinkRealTime.fileSystem.dir

List contents of current folder on target computer

Syntax

```
dir(filesys_object)
dir(filesys_object, folder_name)
dir_info = dir(filesys_object, ___)
```

Description

`dir(filesys_object)` lists the contents of the currently active folder on the target computer.

`dir(filesys_object, folder_name)` lists the contents of folder `folder_name` on the target computer.

`dir_info = dir(filesys_object, ___)` returns the results in a structure array.

Examples

List Contents of Currently Active Folder

List the contents of the currently active folder

```
tg = slrt;
fsys = SimulinkRealTime.fileSystem(tg);
dir(fsys)

    20/6/2011 15:09 <DIR>          0      FDOS
    16/11/2011 14:10 <DIR>          0 $RECYCLE BIN
    30/10/2015 17:38 <DIR>          0   NWR_TMP
    18/8/2006  3:58          45341  KERNEL  SYS
    28/8/2006 18:40          66945  COMMAND COM
    28/3/2013 11:49           1604  AUTOEXEC BAT
    7/11/2011 16:55           207   FDCONFIG SYS
    7/8/2007 12:09          14509  CONFIG  TEL
```

```
25/6/2008 20:18          3066  DEVLOAD  COM
 1/5/2010 14:05          33902  DOSUSB  COM
26/1/2009  3:07          62279  E100BODI COM
21/9/2010 13:00          48123  E1000ODI COM
 7/8/2007  4:42          165262  FTPBIN  EXE
 3/5/1999 15:50          39748   IPXODI  COM
 8/2/2010 20:35          31919  LISTDEVS EXE
30/1/2010  8:34           1394   LPT1USB  SYS
 3/5/1999 15:50          18356   LSL     COM
27/2/2008  8:16           513    NET     CFG
13/6/2002 14:45          3310   ODIPKT30 COM
 7/8/2007 10:16           13    PASSWORD TEL
 9/12/2005 21:06          16536  RTTBOOT  COM
27/2/2008  8:18           236   RUNFTP  BAT
28/8/2008 21:42          1559   SERDRV  SYS
14/6/2002 18:55          17032  TELPASS  EXE
13/6/2002 16:20          1514   TERMIN  COM
 6/3/2010 13:00           7165  USBDISK  SYS
23/1/2010 17:17          36752  USBVIEW  EXE
27/3/2014 11:49           0     DOS     SG
 1/8/2012 15:14          16370  XPCBOOT  COM
27/3/2014 11:49          1140726 XPMTGO  RTB
 6/5/2014 16:28           0     FREEDOS
 6/5/2014 16:45          1276571 XPCKRNL RTB
13/8/2015 17:04          310451 XPCTRACE CSV
17/4/2015 10:53          36503  BOUNCIN1 DLM
30/10/2015 17:04          0     NEW_DATA DAT
```

List Contents of Specific Folder

List the contents of folder 'FDOS'

```
tg = slrt;
fsys = SimulinkRealTime.fileSystem(tg);
dir(fsys, 'FDOS')
```

```
20/6/2011 15:09 <DIR>          0  PACKAGES
20/6/2011 15:09 <DIR>          0  APPINFO
20/6/2011 15:09 <DIR>          0   BIN
20/6/2011 15:09 <DIR>          0   DOC
20/6/2011 15:09 <DIR>          0   HELP
20/6/2011 15:09 <DIR>          0   NLS
20/6/2011 15:09 <DIR>          0   CPI
20/6/2011 15:09 <DIR>          0   TEMP
20/6/2011 15:09          14025  INSTALL  LOG
```

```

15/8/2002 23:59          18353  COPYING
19/5/2006 18:27          26444  COPYING  LIB
 4/9/2006  1:14           8692  POSTINST BAT
 1/9/2006 20:23           3389  POSTSET  BAT
24/1/2004  3:44          11197  CONFIG   SYS

```

Return Contents of Specific Folder as Structure Array

Return the contents of folder 'FDOS' as a structure array.

```

tg = slrt;
fsys = SimulinkRealTime.fileSystem(tg);
dir_info = dir(fsys, 'FDOS')

```

```
dir_info =
```

```
1x14 struct array with fields:
```

```

    date
    time
    isdir
    bytes
    name

```

List one of the items in the array.

```
dir_info(1)
```

```
ns =
```

```

    date: '20/6/2011'
    time: '15:09'
    isdir: 1
    bytes: 0
    name: {'PACKAGES' ''}

```

Input Arguments

filesys_object — Object representing the target computer file system

SimulinkRealTime.fileSystem object

File system object created by using the SimulinkRealTime.fileSystem creation function.

The file system object represents the target computer file system. You work with the target computer file system from the development computer by using file system methods.

Example: `fsys`

Data Types: `struct`

folder_name — Name of a folder on the target computer

character vector

Example: `new_folder`

Data Types: `char`

Output Arguments

dir_info — Structure array containing information about the file or folder being accessed

`struct`

The array consists of the following fields:

- `date` — The last date at which the file or folder was saved.
- `time` — The last time at which the file or folder was saved.
- `isdir` — If 1, the item is a folder. If 0, it is not a folder.
- `bytes` — Size of the file or folder, in bytes.
- `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.

See Also

`dir` | `SimulinkRealTime.fileSystem.mkdir` | `SimulinkRealTime.fileSystem.rmdir`

Introduced in R2014a

SimulinkRealTime.fileSystem.diskinfo

Target computer drive information

Syntax

```
disk_info = diskinfo(filesys_object, drive_name)
```

Description

`disk_info = diskinfo(filesys_object, drive_name)` returns configuration information for the specified drive on the target computer.

Examples

Return Configuration Information About Specified Disk

Return configuration information for the target computer C:\ drive.

```
disk_info = diskinfo(fsys, 'C:\')
```

```
disk_info =
```

```
    DriveLetter: 'C'
```

```
        Label: 'FREEDOS'
```

```
    Reserved: '  '
```

```
    SerialNumber: -857442364
```

```
FirstPhysicalSector: 63
```

```
        FATType: 32
```

```
        FATCount: 2
```

```
    MaxDirEntries: 0
```

```
    BytesPerSector: 512
```

```
    SectorsPerCluster: 64
```

```
    TotalClusters: 1831212
```

```
    BadClusters: 0
```

```
    FreeClusters: 1827665
```

```
        Files: 918
```

```
    FileChains: 919
```

FreeChains: 4
LargestFreeChain: 1827659

Input Arguments

filesystem_object — Object representing the target computer file system

`SimulinkRealTime.fileSystem` object

File system object created by using the `SimulinkRealTime.fileSystem` creation function.

The file system object represents the target computer file system. You work with the target computer file system from the development computer by using file system methods.

Example: `fsys`

Data Types: `struct`

drive_name — Name of the drive to access

character vector

Enclose the drive name in single quotation marks. The drive must exist in the target computer.

Example: `'C:\'`

Data Types: `char`

Output Arguments

disk_info — Structure array containing information about target computer disk drive

`struct`

The disk information includes the drive letter, the internal label of the drive, and the serial number of the disk. It also includes technical information about the disk that a technician can use to debug problems with the disk hardware.

See Also

`SimulinkRealTime.fileSystem.diskSpace`

Introduced in R2014a

SimulinkRealTime.fileSystem.diskspace

Return the free space and total space on the drive, in bytes

Syntax

```
disk_space = diskspace(filesys_object, drive_name)
```

Description

`disk_space = diskspace(filesys_object, drive_name)` returns a structure containing the free space and total space on the drive, in bytes. If a drive with that name does not exist in the target computer, displays an error message.

Examples

Display the Disk Space on the C:\ Drive

Return the free space and total space on the C:\ drive in the target computer.

```
tg = slrt;  
fsys = SimulinkRealTime.fileSystem(tg);  
diskspace(fsys, 'C:\')
```

```
ans =
```

```
    freeDiskSpacebytes: 5.9889e+10  
    totalDiskSpacebytes: 6.0005e+10
```

Input Arguments

filesys_object — Object representing the target computer file system

SimulinkRealTime.fileSystem object

File system object created by using the `SimulinkRealTime.fileSystem` creation function.

The file system object represents the target computer file system. You work with the target computer file system from the development computer by using file system methods.

Example: `fsys`

Data Types: `struct`

drive_name — Name of the drive to access

character vector

Enclose the drive name in single quotation marks. The drive must exist in the target computer.

Example: `'C:\'`

Data Types: `char`

Output Arguments

disk_space — Contains the free space and total space on the drive

`struct`

Returns a structure containing the following fields:

- `freeDiskSpacebytes` — The number of bytes of unused space on the drive.
- `totalDiskSpacebytes` — The total number of bytes on the drive.

See Also

`SimulinkRealTime.fileSystem.diskInfo`

Introduced in R2016a

SimulinkRealTime.fileSystem fclose

Close target computer file

Syntax

```
fclose(filesys_obj, file_id)
```

Arguments

<code>filesys_obj</code>	Name of the <code>SimulinkRealTime.fileSystem</code> file system object
<code>file_id</code>	File identifier of the file to close

Description

From the development computer, `fclose(filesys_obj, file_id)` closes one or more open files in the target computer file system (except standard input, output, and error). The `file_id` argument is the file identifier associated with an open file. You cannot have more than eight files open at the same time in the file system.

Examples

Close the open file identified by the file identifier `h` in the file system object `fsys`:

```
tg = slrt;  
fsys = SimulinkRealTime.fileSystem(tg);  
h = fopen(fsys, 'data.dat', 'w');  
fwrite(fsys, h, 'test')  
fclose(fsys, h)  
h = fopen(fsys, 'data.dat', 'r');  
value = fread(fsys, h);  
char(value)
```

See Also

fclose | File System

Introduced in R2014a

SimulinkRealTime.fileSystem.fileinfo

Target computer file information

Syntax

```
return_value = fileinfo(filesys_obj,file_id)
```

Arguments

<code>filesys_obj</code>	Name of the <code>SimulinkRealTime.fileSystem</code> file system object.
<code>file_id</code>	Identifier of the file for which to get file configuration information.

Description

From the development computer, `return_value = fileinfo(filesys_obj,file_id)` gets file configuration information for the file on the target computer associated with `file_id`.

Examples

Return file configuration information for the target computer file associated with the file identifier `h` in the file system object `fsys`:

```
tg = slrt;  
fsys = SimulinkRealTime.fileSystem(tg);  
h = fopen(fsys, 'data.dat', 'r');  
fileinfo(fsys,h)
```

```
ans =
```

```
FilePos: 0  
AllocatedSize: 32768
```



```
ClusterChains: 1
VolumeSerialNumber: 1082284597
  FullName: 'C:\data.dat'
```

See Also

File System

Introduced in R2014a

SimulinkRealTime.fileSystem.filetable

Information about open files in target computer file system

Syntax

```
return_value = filetable(filesys_obj,file_id)
```

Arguments

<code>filesys_obj</code>	Name of the <code>SimulinkRealTime.fileSystem</code> file system object.
--------------------------	--

Description

Method of `SimulinkRealTime.fileSystem` objects. From the development computer, `return_value = filetable(filesys_obj,file_id)` returns a table of the open files in the target computer file system. You cannot have more than eight files open at the same time in the file system.

Note: Use the `filetable` function only to recover the lost file handle value when MATLAB exits with files still open on the target computer. The function has no other use.

Examples

Return a table of the open files in the target computer file system for the file system object `fsys`:

```
tg = slrt;  
fsys = SimulinkRealTime.fileSystem(tg);  
filetable(fsys)  
ans =
```

Index	Handle	Flags	FilePos	Name
0	00060000	R__	8512	C:\DATA.DAT
1	00080001	R__	0	C:\DATA1.DAT
2	000A0002	R__	8512	C:\DATA2.DAT
3	000C0003	R__	8512	C:\DATA3.DAT
4	001E000S	R__	0	C:\DATA4.DAT

The table returns the open file handles in hexadecimal. To convert a hexadecimal handle to a handle that other `SimulinkRealTime.fileSystem` methods can use, use the MATLAB `hex2dec` function:

```
h1 = hex2dec('001E0001')
h1 =
1966081
```

To close that file, use `SimulinkRealTime.fileSystem.fclose`.

```
fclose(fsys,h1);
```

See Also

File System | `hex2dec`

Introduced in R2014a

SimulinkRealTime.fileSystem.fopen

Open target computer file for reading and writing

Syntax

```
file_id = fopen(file_obj, file_name)
file_id = fopen(file_obj, file_name, permission)
```

Arguments

<code>file_obj</code>	Name of the <code>SimulinkRealTime.fileSystem</code> object.
<code>file_name</code>	Name of the target computer to open, in single quotes
<code>permission</code>	Permission values, one of 'r', 'w', 'a', 'r+', 'w+', or 'a+'.
<code>file_id</code>	Identifier for newly opened file.

The permission values have the following meaning:

- 'r' — Open the file for reading (default). If the file does not exist, the method does not do anything.
- 'w' — Open the file for writing. If the file does not 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 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 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 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 exist, the method creates the file.

Description

From the development computer, `file_id = fopen(file_obj, file_name)` opens the specified file name on the target computer for reading binary data.

`file_id = fopen(file_obj, file_name, permission)` opens the specified file name on the target computer for reading binary data.

You cannot have more than eight files open at the same time in the file system. This method returns the file identifier for the open file in `file_id`. You use `file_id` as the first argument to the other file I/O methods (such as `fclose`, `fread`, and `fwrite`).

There are the following limitations:

- You can have a maximum of eight files open on the target computer at the same time.
- The largest single file that you can create is 4 GB.
- A fully qualified folder name can have a maximum of 248 characters, including the drive letter, colon, and backslash.
- A fully qualified file name can have a maximum of 260 characters. The file part can have 12 characters—a maximum of eight characters for the file name, one character for the period, and a maximum of three characters for the file extension. If the file name is longer than eight characters, the software truncates it to six characters and adds '~1' to the end of the file name.

Examples

Open the file `data.dat` in the target computer file system object `fsys` and read the file using the resulting file handle:

```
tg = slrt;  
fsys = SimulinkRealTime.fileSystem(tg);  
h = fopen(fsys, 'data.dat');
```

```
ans =
```

```
2883584
```

```
d = fread(fsys,h);
```

See Also

File System | `fopen`

Introduced in R2014a

SimulinkRealTime.fileSystem.fread

Read open target computer file

Syntax

```
data = fread(file_obj,file_id)
data = fread(file_obj,file_id,offset,numbytes)
```

Arguments

<code>file_obj</code>	Name of the <code>SimulinkRealTime.fileSystem</code> object.
<code>file_id</code>	File identifier of the file to read.
<code>numbytes</code>	Maximum number of bytes <code>fread</code> can read.
<code>offset</code>	The position, measured from the beginning of the file, from which <code>fread</code> can start to read.
<code>data</code>	Matrix containing the binary data read.

Description

`data = fread(file_obj,file_id)` reads binary data from the file on the target computer and writes it into matrix `data`. The `file_id` argument is the file identifier associated with an open file.

`data = fread(file_obj,file_id,offset,numbytes)` reads `numbytes` bytes from `file_id` starting from position `offset` and writes the block into matrix `data`.

To get a count of the total number of bytes read into `data`, use the following:

```
count = length(data);
```

If `numbytes` bytes are not available, `length(data)` can be less than `numbytes`. `length(data)` is zero if `fread` is positioned at the end of the file.

Examples

Open the file `data.dat` in the target computer file system object `fsys` and read the file using the resulting file handle:

```
tg = slrt;  
fsys = SimulinkRealTime.fileSystem(tg);  
h = fopen(fsys, 'data.dat')  
d = fread(fsys,h);
```

This function reads the file `data.dat` and stores the contents of the file to `d`. This content is in the Simulink Real-Time file format.

See Also

File System | `fread`

Introduced in R2014a

SimulinkRealTime.fileSystem.fwrite

Write binary data to open target computer file

Syntax

```
fwrite(file_obj,file_id,data)
```

Arguments

<code>file_obj</code>	Name of the <code>SimulinkRealTime.fileSystem</code> object.
<code>file_id</code>	File identifier of the file to write.
<code>data</code>	Elements of matrix <code>data</code> to write to the specified file.

Description

From the development computer, `fwrite(file_obj,file_id,data)` writes the elements of matrix `data` to the file identified by `file_id`. The data is written to the file in column order. The `file_id` argument is the file identifier associated with an open file. `fwrite` requires that the file is open with write permission.

Examples

Open the file `data.dat` in the target computer file system object `fsys` and write the file using the resulting file handle:

```
tg = slrt;  
fsys = SimulinkRealTime.fileSystem(tg);  
h = fopen(fsys,'data.dat','w');  
fwrite(fsys,h,magic(5));
```

This command writes the elements of matrix `magic(5)` to the file handle `h`. This content is written in column order.

See Also

File System | fwrite

Introduced in R2014a

SimulinkRealTime.fileSystem.getfilesize

Size of file on target computer

Syntax

```
file_size = getfilesize(file_obj,file_id)
```

Arguments

<code>file_obj</code>	Name of the <code>SimulinkRealTime.fileSystem</code> object
<code>file_id</code>	File identifier of the file being sized
<code>file_size</code>	Number of bytes in the file

Description

From the development computer, `file_size = getfilesize(file_obj,file_id)` gets the size (in bytes) of the file identified by the `file_id` file identifier on the target computer file system. Use the Simulink Real-Time file object method `fopen` to open the file system object.

Examples

Get the size of the file identifier `h` for the file system object `fsys`:

```
tg = slrt;  
fsys = SimulinkRealTime.fileSystem(tg);  
getfilesize(fsys,h)
```

See Also

File System

Introduced in R2014a

SimulinkRealTime.fileSystem.mkdir

Create folder on target computer

Syntax

```
mkdir(file_obj,dir_name)
```

Arguments

<code>file_obj</code>	Name of the <code>SimulinkRealTime.fileSystem</code> object.
<code>dir_name</code>	Name of the folder to be created.

A fully qualified folder name can have a maximum of 248 characters, including the drive letter, colon, and backslash.

Description

From the development computer, `mkdir(file_obj,dir_name)` makes a new folder in the current folder on the target computer file system.

Examples

Create a folder, `logs`, in the target computer file system object `fsys`:

```
tg = slrt;  
fsys = SimulinkRealTime.fileSystem(tg);  
mkdir(fsys, 'logs')
```

See Also

File System | `mkdir`

Introduced in R2014a

SimulinkRealTime.fileSystem.pwd

Path to currently active folder on target computer

Syntax

```
active_folder = pwd(file_obj)
```

Arguments

file_obj Name of the `SimulinkRealTime.fileSystem` object.
active_folder Path to the currently active folder on the target computer.

Description

Called from the development computer, `active_folder = pwd(file_obj)` returns the path to the currently active folder on the target computer. Unless `cd(file_obj, target_computer_dir)` has been called, the currently active folder is the top folder of the boot drive, usually `C:\`.

Examples

Return the currently active folder for the file system object `fsys`:

```
tg = slrt;  
fsys = SimulinkRealTime.fileSystem(tg);  
pwd(fsys)
```

See Also

File System | `pwd`

Introduced in R2014a

SimulinkRealTime.fileSystem.removefile

Remove file from target computer

Syntax

```
removefile(file_obj,file_name)
```

Arguments

<code>file_name</code>	Name of the file to remove from the target computer file system.
<code>file_obj</code>	Name of the <code>SimulinkRealTime.fileSystem</code> object.

Description

Called from the development computer, `removefile(file_obj,file_name)` removes a file from the target computer file system.

Note: You cannot recover this file once you remove it

Examples

Remove the `data2.dat` file from the target computer file system `fsys`:

```
tg = slrt;  
fsys = SimulinkRealTime.fileSystem(tg);  
removefile(fsys,'data2.dat')
```

See Also

File System

Introduced in R2014a

SimulinkRealTime.fileSystem.rename

Rename a file or folder in the target computer disk drive

Syntax

```
rename(filesys_object, 'old_name', 'new_name')
```

Description

`rename(filesys_object, 'old_name', 'new_name')` renames a file or folder in the target computer disk drive. If the file is open or does not exist, the function displays an error message.

Examples

Rename a File in the Current Folder

Renames the file `old_data.dat` to `new_data.dat` in the current folder.

```
tg=slrt;  
fsys=SimulinkRealTime.fileSystem(tg);  
dir(fsys);
```

```
30/10/2015 17:29          0 OLD_DATA  DAT
```

If `old_data.dat` is open, close it with `SimulinkRealTime.fileSystem.fclose`.

```
rename(fsys, 'old_data.dat', 'new_data.dat');  
dir(fsys);
```

```
30/10/2015 17:29          0 NEW_DATA  DAT
```

Rename a File in a Folder

Renames the file `C:\old_temp\old_data.dat` to `C:\old_temp\new_data.dat`.

```
tg=slrt;  
fsys=SimulinkRealTime.fileSystem(tg);
```

```
dir(fsys, 'C:\old_temp');
```

```
30/10/2015 17:29          0 OLD_DATA  DAT
```

If `old_data.dat` is open, close it with `SimulinkRealTime.fileSystem fclose`.

```
rename(fsys, 'C:\old_temp\old_data.dat', ...
       'C:\old_temp\new_data.dat');
```

```
dir(fsys, 'C:\old_temp');
```

```
30/10/2015 17:29          0 NEW_DATA  DAT
```

Move a File from One Folder to Another

Moves the file `C:\old_temp\new_data.dat` to `C:\new_temp\new_data.dat` by renaming the folder part of the path.

```
tg=slrt;
fsys=SimulinkRealTime.fileSystem(tg);
dir(fsys, 'C:\old_temp');
```

```
30/10/2015 17:29          0 NEW_DATA  DAT
```

If `new_data.dat` is open, close it with `SimulinkRealTime.fileSystem fclose`. If `C:\new_temp` does not exist, create it by using `SimulinkRealTime.fileSystem mkdir`.

```
rename(fsys, 'C:\old_temp\new_data.dat', ...
       'C:\new_temp\new_data.dat');
```

```
dir(fsys, 'C:\new_temp');
```

```
30/10/2015 17:29          0 NEW_DATA  DAT
```

Input Arguments

filesys_object — Object representing the target computer file system

`SimulinkRealTime.fileSystem` object

File system object created by using the `SimulinkRealTime.fileSystem` creation function.

The file system object represents the target computer file system. You work with the target computer file system from the development computer by using file system methods.

Example: `fsys`

Data Types: `struct`

old_name — Old name of file or folder

character vector

The old name of the file or folder can be a name relative to the current folder or a fully qualified path. Enclose the name in single quotation marks.

Example: `'old_data.dat'`, `'C:\old_temp\old_data.dat'`

Data Types: `char`

new_name — New name of file or folder

character vector

The new name of the file or folder can be a name relative to the current folder or a fully qualified path. Enclose the name in single quotation marks. If you are moving a file to a different folder, the folder must exist.

Example: `'new_data.dat'`, `'C:\new_temp\new_data.dat'`

Data Types: `char`

See Also

File System

Introduced in R2016a

SimulinkRealTime.fileSystem.rmdir

Remove empty folder from target computer

Syntax

```
rmdir(file_obj,dir_name)
```

Arguments

<code>dir_name</code>	Name of the folder to remove from the target computer file system.
<code>file_obj</code>	Name of the <code>SimulinkRealTime.fileSystem</code> object.

Description

Called from the development computer, `rmdir(file_obj,dir_name)` removes an empty folder from the target computer file system. If the folder contains a file or folder, the function prints an error message.

Note: You cannot recover this folder once you remove it.

Examples

Remove the `data2dir.dat` folder from the target computer file system `fsys`:

```
tg = slrt;  
fsys = SimulinkRealTime.fileSystem(tg);  
rmdir(fsys,'data2dir.dat')
```

See Also

File System

Introduced in R2014a

SimulinkRealTime.fileSystem.selectdrive

Select target computer drive

Syntax

```
selectdrive(filesys_object,drive_name)
```

Description

`selectdrive(filesys_object,drive_name)` sets the currently active drive of the target computer to the specified character vector. If a drive with that name does not exist in the target computer, the function displays an error message.

Examples

Select the C:\ Drive

Select the C:\ drive in the target computer.

```
tg = slrt;  
fsys = SimulinkRealTime.fileSystem(tg);  
selectdrive(fsys,'C:\')
```

Input Arguments

filesys_object — Object representing the target computer file system

`SimulinkRealTime.fileSystem` object

File system object created by using the `SimulinkRealTime.fileSystem` creation function.

The file system object represents the target computer file system. You work with the target computer file system from the development computer by using file system methods.

Example: fsys

Data Types: struct

drive_name — Name of the drive to access

character vector

Enclose the drive name in single quotation marks. The drive must exist in the target computer.

Example: 'C:\'

Data Types: char

See Also

File System

Introduced in R2016a

Real-Time Application

Represent real-time application and target computer status

Description

Object represents currently loaded real-time application and target computer status.

Object provides access to methods and properties that do the following:

- Start and stop the real-time application.
- Read and set parameters.
- Monitor signals.
- Retrieve status information about the target computer.
- Restart the target computer.
- Load and unload the real-time application.

Function names are case sensitive. Type the entire name. Property names are not case sensitive. You do not need to type the entire name, as long as the characters you do type are unique for the property.

Some of the object properties and functions can be invoked from the target computer command line when the real-time application has been loaded.

Create Object

`SimulinkRealTime.target`

Properties

Real-Time Application Properties

Properties of real-time application and target computer

Object Functions

`SimulinkRealTime.target.ping`

Test communication between development and target computers

`SimulinkRealTime.target.reboot`

Restart target computer

<code>SimulinkRealTime.target.load</code>	Download real-time application to target computer
<code>SimulinkRealTime.target.unload</code>	Remove real-time application from target computer
<code>SimulinkRealTime.target.close</code>	Close connection between development and target computers
<code>SimulinkRealTime.target.start</code>	Start execution of real-time application on target computer
<code>SimulinkRealTime.target.stop</code>	Stop execution of real-time application on target computer
<code>SimulinkRealTime.target.addscope</code>	Create a scope of specified type
<code>SimulinkRealTime.target.getscope</code>	Return scope identified by scope number
<code>SimulinkRealTime.target.remscope</code>	Remove scope from target computer
<code>SimulinkRealTime.target.getlog</code>	Portion of output logs from target object
<code>SimulinkRealTime.target.getsignal</code>	Value of signal
<code>SimulinkRealTime.target.getsignalid</code>	Signal index from signal hierarchical name
<code>SimulinkRealTime.- target.getsignalidsfromlabel</code>	Vector of signal indices
<code>SimulinkRealTime.target.getsignallabel</code>	Signal label for signal index
<code>SimulinkRealTime.target.getsignalname</code>	Signal name from index list
<code>SimulinkRealTime.target.getparam</code>	Read value of observable parameter in real-time application
<code>SimulinkRealTime.target.setparam</code>	Change value of tunable parameter in real-time application
<code>SimulinkRealTime.target.getparamid</code>	Parameter index from parameter hierarchical name
<code>SimulinkRealTime.target.getparamname</code>	Block path and parameter name from parameter index
<code>SimulinkRealTime.target.loadparamset</code>	Restore parameter values saved in specified file
<code>SimulinkRealTime.target.saveparamset</code>	Save real-time application parameter values

Examples

Build and run real-time application

Build and download `xpcosc`, execute real-time application in external mode.

Open, build, and download real-time application

```
ex_model = 'xpcosc';  
open_system(ex_model);  
ex_scope = [ex_model '/Scope'];  
open_system(ex_scope)  
rtwbuild(ex_model);  
tg = SimulinkRealTime.target
```

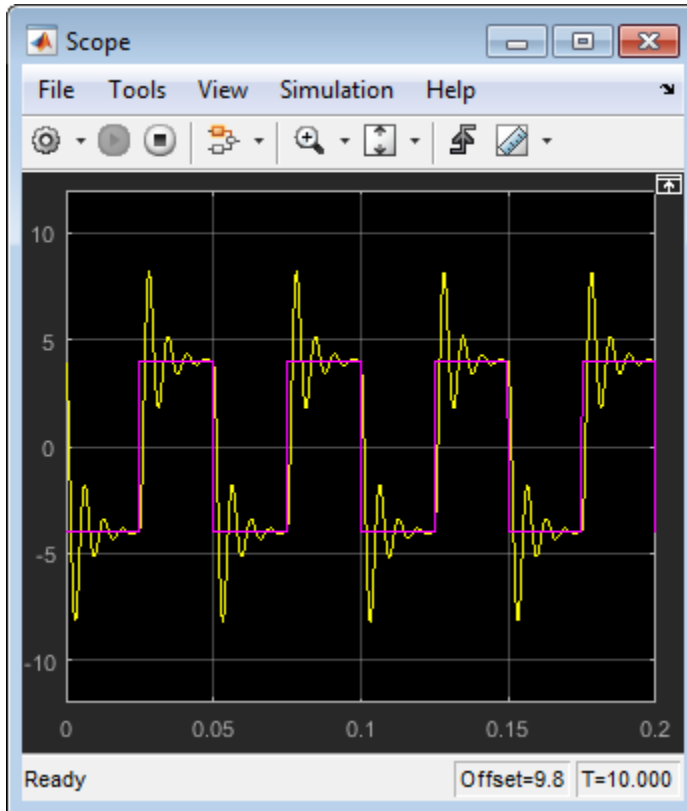
```
Target: TargetPC1  
Connected           = Yes  
Application         = xpcosc  
Mode                = Real-Time Single-Tasking  
Status              = stopped  
CPUOverload         = none  
  
ExecTime            = 0.0000  
SessionTime         = 794.4953  
StopTime            = 0.200000  
SampleTime          = 0.000250  
AvgTET              = NaN  
MinTET              = Inf  
MaxTET              = 0.000000  
ViewMode            = 0  
  
TimeLog             = Vector(0)  
StateLog            = Matrix (0 x 2)  
OutputLog           = Matrix (0 x 2)  
TETLog              = Vector(0)  
MaxLogSamples       = 16666  
NumLogWraps         = 0  
LogMode             = Normal  
  
Scopes              = No Scopes defined  
NumSignals          = 7  
ShowSignals         = off  
  
NumParameters       = 7  
ShowParameters      = off
```

Prepare and run simulation in external mode for 10 seconds.

```
tg.StopTime = 10;  
set_param(ex_model, 'SimulationMode', 'External');  
set_param(ex_model, 'SimulationCommand', 'Connect');  
set_param(ex_model, 'SimulationCommand', 'Start');  
pause(10);
```

```
set_param(ex_model, 'SimulationCommand', 'Stop');
set_param(ex_model, 'SimulationCommand', 'Disconnect');
```

The output looks like this:



Unload real-time application

```
unload(tg)
```

Target: TargetPC1

Connected = Yes

Application = loader

See Also

“Target Computer Commands”

More About

- “Blocks Whose Outputs Depend on Inherited Sample Time”

Introduced in R2014a

Real-Time Application Properties

Properties of real-time application and target computer

Description

Provides access to the properties of the real-time application and the target computer.

To get the value of a readable target object property from a target object:

```
value = target_object.property_name
```

For example, to get the `CommunicationTimeOut` of the target object:

```
target_object = slrt;
value = target_object.CommunicationTimeOut
```

To set the value of a writable target object property from a target object:

```
target_object.property_name = new_value
```

For example, to set the `CommunicationTimeOut` of the target object:

```
target_object = slrt;
target_object.CommunicationTimeOut = 10
```

At the target computer command line, you can set the target object properties `stoptime`, `sampletime`, and writable parameters.

```
stoptime = floating_point_number
sampletime = floating_point_number
setpar parameter_index = parameter_value
```

Target Computer

Application — Name of real-time application

'loader' | character vector

This property is read only.

Name of real-time application running on target computer, specified as a character vector. This is the name of the Simulink model from which the application was built. When the target computer starts, this value is 'loader'.

CommunicationTimeout — Communication timeout between development and target computers

5 (default) | seconds

Communication timeout between the development and target computers, specified in seconds.

Connected — Communication status between development and target computers

'No' (default) | 'Yes'

This property is read only.

Communication status between the development and target computers, specified as character vector.

CPUoverload — CPU status for overload

'none' (default) | 'detected'

This property is read only.

CPU status for overload, specified as character vector. If the real-time application requires more CPU time than the model sample time, the kernel changes this value from 'none' to 'detected' and stops the current run. To keep this status 'none' you must use a faster processor or specify a larger sample time.

Mode — Execution mode of the real time application

'Real-Time Singletasking' (default) | 'Real-Time Multitasking'

This property is read only.

Execution mode of the real time application on the target computer, specified as a character vector. Parameter settings determine the execution mode during Simulink Coder code generation.

SessionTime — Time since kernel started running on target computer

seconds

This property is read only.

Time since the kernel started running on the target computer, specified in seconds. This time is also the elapsed time since you started the target computer.

Real-Time Execution

AvgTET — Average task execution time

seconds

This property is read only.

Average task execution time, specified in seconds.

Task Execution Time (TET) measures how long it takes the kernel to run for one base-rate time step. For a multirate model, use the profiler to find out what the execution time is for each rate.

Task execution time is nearly constant, with minor deviations due to cache, memory access, interrupt latency, and multirate model execution.

The TET includes:

- Complete I/O latency.
- Data logging for output, state, and TET, and the data captured in scopes.
- Time spent executing tasks related to asynchronous interrupts while the real time task is running.
- Parameter updating latency. This latency is incurred if the **Double buffer parameter changes** parameter is set in the **Simulink Real-Time Options** node of the model Configuration Parameters dialog box.

The TET is not the only consideration in determining the minimum achievable sample time. Other considerations are:

- Time required to measure TET.
- Interrupt latency required to schedule and run one step of the model.

ExecTime — Execution time of real-time application

seconds

This property is read only.

Execution time of real-time application since your real-time application started running, specified in seconds. When the real-time application stops, the kernel displays the total execution time.

MaxTET — Maximum task execution time

seconds

This property is read only.

Maximum task execution time, specified in seconds. Corresponds to the slowest time (longest measured time) required to update model equations and post outputs.

MinTET — Minimum task execution time

seconds

This property is read only.

Minimum task execution time, specified in seconds. Corresponds to the fastest time (smallest measured time) required to update model equations and post outputs.

SampleTime — Time between samples (step size)

seconds

Time between samples (step size), in seconds, for updating the model equations and posting the outputs.

Note: Some blocks produce incorrect results when you change their sample time at run time. If you include such blocks in your model, the software displays a warning message during model build. To avoid incorrect results, change the sample time in the original model, and then rebuild and download the model.

See “Limits on Sample Time”.

Status — Execution status of real-time application

'stopped' (default) | 'running'

This property is read only.

Execution status of real-time application, specified as character vector.

StopTime — Time when real-time application stops running

seconds | 'Inf'

Time when the real-time application stops running, specified in seconds or as character vector. The initial value is set in the **Solver** pane of the Configuration Parameters dialog box.

When the `ExecTime` reaches `StopTime`, the application stops running. If you specify the special value `'Inf'`, the real-time application runs until you manually stop it or restart the target computer.

TETLog — Storage in the MATLAB workspace for task execution time vector

vector of double

This property is read only.

Storage in the MATLAB workspace for task execution time vector, specified as a vector of double.

Signal Visualization

LogMode — Controls which data points are logged

'Normal' (default) | double

Controls which data points are logged, as specified by the keyword `'Normal'` or a double.

- `'Normal'` — Indicates time-equidistant logging. Logs a data point at every time interval.
- `Double` — Indicates value-equidistant logging. Logs a data point only when an output signal from the `OutputLog` changes by the specified difference in signal value (increment).

MaxLogSamples — Maximum number of samples for each logged signal

unsigned integer

This property is read only.

Maximum number of samples for each logged signal, specified as an unsigned integer.

NumLogWraps — Number of times the circular data logging buffer wraps

unsigned integer

This property is read only.

Number of times the circular data logging buffer wraps, specified as an unsigned integer. The buffer wraps each time the number of samples exceeds `MaxLogSamples`.

NumSignals — Number of observable signals

unsigned integer

This property is read only.

Number of observable signals in Simulink model, specified as an unsigned integer. Nonobservable signals are not included in this value.

Note:

- Signal access by signal index will be removed in a future release. Access signals by signal name instead.
 - This parameter will be removed in a future release.
-

OutputLog — Storage in MATLAB workspace for output or Y-vector
matrix

This property is read only.

Storage in MATLAB workspace for output or Y-vector, specified as a matrix.

Scopes — List of index numbers, one per scope
vector of unsigned integer

This property is read only.

List of index numbers, one per scope, specified as a vector of unsigned integers.

ShowSignals — Flag set to display the list of signals
'off' (default) | 'on'

Flag set to view the list of signals from your Simulink model, specified as character vector. MATLAB displays the signal list when you display the properties for a target object.

Signals — List of observable signals
vector of structures

This property is read only.

List of observable signals, specified as a vector containing the following values for each signal:

- Index — ID used to access the signal.

- Value — Value of the signal.
- Type — Data type of the signal.
- Block name— Hierarchical name of the Simulink block that the signal comes from.
- Label — Label that you have assigned to this signal.

This list is visible only when `ShowSignals` is set to 'on'.

StateLog — Storage in MATLAB workspace for state or X-vector
matrix

This property is read only.

Storage in MATLAB workspace for state or X-vector, specified as a matrix.

TimeLog — Storage in the MATLAB workspace for time or T-vector
vector of double

This property is read only.

Storage in the MATLAB workspace for time or T-vector, specified as a vector of double.

Parameter Tuning

NumParameters — Number of tunable parameters
unsigned integer

This property is read only.

Number of tunable parameters in Simulink model, specified as an unsigned integer. Nontunable (nonobservable) parameters are not included in this value.

Note:

- Parameter access by parameter index will be removed in a future release. Access parameters by parameter name instead.
 - This parameter will be removed in a future release.
-

Parameters — List of tunable parameters
vector of structures

This property is read only.

List of tunable parameters, specified as a vector containing the following values for each parameter:

- Value — Value of the parameter in a Simulink block. If the parameter is a structure, the value is displayed with vector brackets.
- Type — Data type of the parameter.

Note: Simulink Real-Time does not support parameters of multiword data types.

- Size — Size of the parameter. For example, scalar, 1-by-2 vector, or 2-by-3 matrix, structure.
- Parameter name — Name of the parameter in a Simulink block.

If the parameter is a field of a structure, the name is displayed in the form `structname.fieldname`.

- Block name — If the parameter is a block parameter, the block name is the hierarchical name of the Simulink block containing the parameter. If the parameter is a MATLAB variable that provides the value for a block parameter, the block name is the empty character vector.

This list is visible only when `ShowParameters` is set to `'on'`.

ShowParameters — Flag set to display the list of parameters

`'off'` (default) | `'on'`

Flag set to view the list of parameters from your Simulink model, specified as character vector. MATLAB displays the parameter list when you display the properties for a target object.

See Also

“Target Computer Commands” | Real-Time Application
| `SimulinkRealTime.target.getsignalid` |
`SimulinkRealTime.utils.minimumSampleTime`

More About

- “Nonobservable Signals”

- “Nonobservable Parameters”
- “Limits on Sample Time”

Introduced in R2014a

SimulinkRealTime.target

Create object representing real-time application on target computer

Syntax

```
target_object = SimulinkRealTime.target  
target_object = SimulinkRealTime.target(target_name)
```

Description

`target_object = SimulinkRealTime.target` constructs a target object representing the default target computer.

When MATLAB evaluates the return value on the development computer, it attempts to connect to the target computer. If the attempt succeeds, MATLAB prints `Connected = Yes`, followed by the status of the real-time application running on the target computer. If the attempt fails, MATLAB waits until the connection times out, and then prints `Connected = No`. To avoid the timeout delay, verify that the target computer is operational and connected to the development computer, or suppress output with a terminating semicolon.

`target_object = SimulinkRealTime.target(target_name)` constructs a target object representing the target computer designated by `target_name`.

Examples

Default Target Computer

Create a target object that communicates with the default target computer. Report the status of the default target computer. In this case, the target computer is connected to the development computer and is executing the loader.

```
target_object = SimulinkRealTime.target
```

```
Target: TargetPC1  
Connected = Yes
```

```
Application = loader
```

Specific Target Computer

Create a target object that communicates with target computer `TargetPC1`. Report the status of the target computer. In this case, the target computer is not connected to the development computer.

```
target_object = SimulinkRealTime.target('TargetPC1')
```

```
Target: TargetPC1
Connected = No
```

Input Arguments

target_name — Name assigned to target computer

character vector

Example: 'TargetPC1'

Data Types: char

Output Arguments

target_object — Object representing target computer

SimulinkRealTime.target object

Object that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required Ethernet link settings.

Example: tg

Data Types: struct

See Also

Real-Time Application | Real-Time Application Properties | `s1rt` | Target Settings Properties

Introduced in R2014a

SimulinkRealTime.target.addscope

Create a scope of specified type

Syntax

```
scope_object = addscope(target_object)
scope_object = addscope(target_object, scope_type, scope_number)
scope_object_vector = addscope(target_object, scope_type,
scope_number_vector)
```

Description

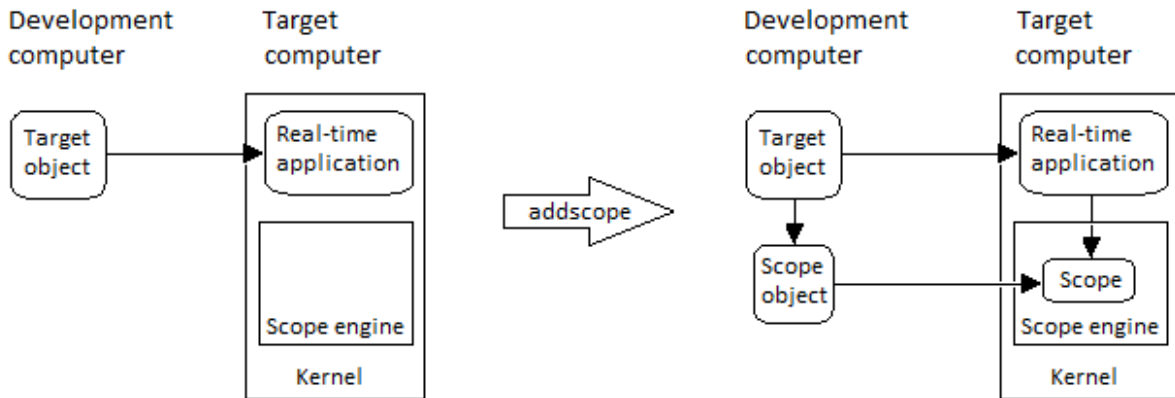
`scope_object = addscope(target_object)` creates on the target computer a host scope and assigns as its scope number the next available integer in the target object property `Scopes`. It returns the object representing this scope.

`scope_object = addscope(target_object, scope_type, scope_number)` creates on the target computer a scope of the given type with the given scope number. It returns the object representing this scope.

`scope_object_vector = addscope(target_object, scope_type, scope_number_vector)` creates on the target computer a set of scopes of the given type with the given scope numbers. It returns a vector of objects representing these scopes.

`addscope` updates the target object property `Scopes`. If the result is not assigned to a MATLAB variable, the scope object properties are listed in the Command Window.

The Simulink Real-Time product supports nine target scopes, eight file scopes, and as many host scopes as the target computer resources can support. If you try to add a scope with the same index as an existing scope, the result is an error.



At the target computer command line, you can add a single target scope:

```
addscope
addscope scope_number
```

Examples

Create default scope with default number

Create a default (host) scope with the default (next available) number and assign it to `sc1`

```
tg = slrt;
sc1 = addscope(tg)
```

```
sc1 =
```

```
Simulink Real-Time Scope
  Application      = xpcosc
  ScopeId         = 1
  Status          = Interrupted
  Type            = Host
  NumSamples      = 250
  NumPrePostSamples = 0
  Decimation      = 1
  TriggerMode     = FreeRun
```

```
TriggerSignal      = -1
TriggerLevel      = 0.000000
TriggerSlope      = Either
TriggerScope      = 1
TriggerSample     = 0
StartTime         = -1.000000
Data              = Matrix (250 x 0)
Time              = Matrix (250 x 1)
Signals           = no Signals defined
```

Create file scope number 2

Create a file scope with number 2 and assign it to `sc2`.

```
tg = slrt;
sc2 = addscope(tg, 'file', 2)
```

```
sc2 =
```

```
Simulink Real-Time Scope
Application        = xpcosc
ScopeId           = 2
Status            = Interrupted
Type              = File
NumSamples        = 250
NumPrePostSamples = 0
Decimation        = 1
TriggerMode       = FreeRun
TriggerSignal     = -1
TriggerLevel      = 0.000000
TriggerSlope     = Either
TriggerScope     = 2
TriggerSample     = 0
FileName          = unset
WriteMode         = Lazy
WriteSize         = 512
AutoRestart       = off
DynamicFileName   = off
MaxWriteFileSize  = 536870912
Signals           = no Signals defined
```

Create vector of target scopes numbers 3 and 4

Create two target scopes 3 and 4 using a vector of scope numbers and assign the scope objects to variable `scvector`.

```
tg = slrt;
scope_object_vector = addscope(tg, 'target', [3, 4])

scope_object_vector =

Simulink Real-Time Scope
  Application      = xpcos  ScopeId      = 3
  Status          = Interrupted
  Type            = Target
  NumSamples      = 250
  NumPrePostSamples = 0
  Decimation      = 1
  TriggerMode     = FreeRun
  TriggerSignal   = -1
  TriggerLevel    = 0.000000
  TriggerSlope    = Either
  TriggerScope    = 3
  TriggerSample   = 0
  DisplayMode     = Redraw (Graphical)
  YLimit          = Auto
  Grid            = on
  Signals         = no Signals defined

Simulink Real-Time Scope
  Application      = xpcosc
  ScopeId         = 4
  Status          = Interrupted
  Type            = Target
  NumSamples      = 250
  NumPrePostSamples = 0
  Decimation      = 1
  TriggerMode     = FreeRun
  TriggerSignal   = -1
  TriggerLevel    = 0.000000
  TriggerSlope    = Either
  TriggerScope    = 4
  TriggerSample   = 0
  DisplayMode     = Redraw (Graphical)
  YLimit          = Auto
  Grid            = on
```

Signals = no Signals defined

Input Arguments

target_object — Object representing target computer

SimulinkRealTime.target object

Object that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required Ethernet link settings.

Example: tg

Data Types: struct

scope_type — Type of scope to create

'host' (default) | 'target' | 'file'

Type of scope to create, as a character vector. This argument is optional. The default value is 'host'.

scope_number — New scope number

unsigned integer

New scope number. This argument is optional. The default value is the next available integer in the target object property `Scopes`.

If you enter the scope number for an existing scope object, the result is an error.

Example: 1

scope_number_vector — Vector of new scope numbers

unsigned integer vector

Vector of new scope numbers. If you enter the scope number for an existing scope object, the result is an error.

Example: [2, 3]

Output Arguments

scope_object — Object representing newly created scope

object

Object representing the newly created scope

scope_object_vector — Vector of objects representing newly created scope
object

Vector containing objects representing the newly created scope

See Also

“Target Computer Commands” | Real-Time Application | Real-Time Application Properties | `SimulinkRealTime.target.getscope` | `SimulinkRealTime.target.remscope`

Introduced in R2014a

SimulinkRealTime.target.close

Close connection between development and target computers

Syntax

```
status_char_vector = close(target_object)
```

Description

`status_char_vector = close(target_object)` closes the connection between the development computer and a target computer. The target object and other associated objects are still valid, and will automatically connect to the target computer the next time they are accessed.

Examples

Close Communication with Target Computer 'TargetPC1'

Access target computer 'TargetPC1' and close the connection.

Get a target object for target computer 'TargetPC1'

```
tg = SimulinkRealTime.target('TargetPC1')
```

```
Target: TargetPC1
  Connected           = Yes
  Application         = loader
```

Close the connection

```
close(tg)
```

```
ans =
```

Communication is closed

Input Arguments

target_object — Object representing target computer

SimulinkRealTime.target object

Object that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required Ethernet link settings.

Example: tg

Data Types: struct

Output Arguments

status_char_vector — Report results of attempt to close communication

'Communication is closed'

Returns literal character vector on every call, unless `close` failed.

See Also

Real-Time Application | Real-Time Application Properties |
SimulinkRealTime.target | SimulinkRealTime.target.reboot

Introduced in R2014a

SimulinkRealTime.target.getlog

Portion of output logs from target object

Syntax

```
log = getlog(target_object, log_name)
log = getlog(target_object, log_name, first_point)
log = getlog(target_object, log_name, first_point, number_samples)
log = getlog(target_object, log_name, first_point, number_samples,
decimation)
```

Description

`log = getlog(target_object, log_name)` returns all the samples from a log of type `log_name`, starting from the first point without decimation.

`log = getlog(target_object, log_name, first_point)` returns the sample at `first_point` from a log of type `log_name`.

`log = getlog(target_object, log_name, first_point, number_samples)` returns `number_samples` samples from a log of type `log_name`, starting from `first_point` without decimation.

`log = getlog(target_object, log_name, first_point, number_samples, decimation)` returns `number_samples` samples from a log of type `log_name`, starting from `first_point`, with decimation `decimation`.

Examples

Retrieve All Values

Read the TimeLog and OutputLog samples from model `xpcosc` using the default settings. Plot the results.

Read TimeLog and OutputLog samples

```
tg = slrt;  
timelog = getlog(tg, 'TimeLog');  
outputlog = getlog(tg, 'OutputLog');
```

Plot the data

```
plot(timelog, outputlog);
```

Retrieve 10 Values Starting from 5

Read 10 samples starting from 5 of TimeLog and OutputLog

Read 5 TimeLog samples

```
tg = slrt;  
timelog = getlog(tg, 'TimeLog', 5, 10)
```

```
timelog =
```

```
0.0010  
0.0013  
0.0015  
0.0018  
0.0020  
0.0023  
0.0025  
0.0027  
0.0030  
0.0033
```

Read 10 OutputLog samples

```
outputlog = getlog(tg, 'OutputLog', 5, 10)
```

```
outputlog =
```

```
-1.6200  -4.0000  
-2.3450  -4.0000  
-3.0990  -4.0000  
-3.8345  -4.0000  
-4.5098  -4.0000  
-5.0907  -4.0000  
-5.5518  -4.0000  
-5.8772  -4.0000  
-6.0606  -4.0000
```

```
-6.1046 -4.0000
```

Plot the data

```
plot(timelog, outputlog);
```

Retrieve 10 values starting from 5 with decimation 2

Read 10 samples at decimation 2 starting from 5 of TimeLog and OutputLog

Read 5 TimeLog samples

```
tg = slrt;  
timelog = getlog(tg, 'TimeLog', 5, 10, 2)
```

```
timelog =
```

```
0.0010  
0.0015  
0.0020  
0.0025  
0.0030  
0.0035  
0.0040  
0.0045  
0.0050  
0.0055
```

Read 10 OutputLog samples

```
outputlog = getlog(tg, 'OutputLog', 5, 10, 2)
```

```
-1.6200 -4.0000  
-3.0990 -4.0000  
-4.5098 -4.0000  
-5.5518 -4.0000  
-6.0606 -4.0000  
-6.0199 -4.0000  
-5.5384 -4.0000  
-4.8028 -4.0000  
-4.0224 -4.0000  
-3.3784 -4.0000
```

Plot the data

```
plot(timelog, outputlog);
```

Retrieve 1 value

Read 1 sample starting from sample 8 of TimeLog and OutputLog

Read 5 TimeLog samples

```
tg = slrt;
timelog = getlog(tg, 'TimeLog', 8)
```

```
timelog =
```

```
    0.0018
```

Read 10 OutputLog samples

```
outputlog = getlog(tg, 'OutputLog', 8)
```

```
outputlog =
```

```
   -3.8345   -4.0000
```

Input Arguments

target_object — Object representing target computer

SimulinkRealTime.target object

Object that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required Ethernet link settings.

Example: tg

Data Types: struct

log_name — Selects information type to retrieve

'TimeLog' | 'StateLog' | 'OutputLog' | 'TETLog'

- TimeLog — Timestamps for each logged value
- StateLog — Discrete and continuous state of blocks
- OutputLog — Value of root-level output blocks

- **TETLog** — Task execution times (TET)

Example: 'TimeLog'

Data Types: char

first_point — Sample from which to start retrieving data

1 (default) | positive integer

If specified without `number_samples`, this parameter returns only the value at `first_point`.

Example: 10

number_samples — Number of samples to retrieve

all points in log (default) | positive integer

Number of samples to retrieve starting with `first_point`, after decimation.

Example: 10

decimation — Select every decimationth value

1 (default) | positive integer

1 returns all sample points. `n` returns every `n`th sample point. Must be used with `first_point` and `number_samples`.

Example: 2

Output Arguments

log — User-defined MATLAB variable

matrix

Variable receives the log entries as a matrix

More About

- “Set Configuration Parameters”

Introduced in R2014a

SimulinkRealTime.target.getparam

Read value of observable parameter in real-time application

Syntax

```
value = getparam(target_object, parameter_block_name,  
parameter_name)
```

```
value = getparam(target_object, parameter_name)
```

```
value = getparam(target_object, parameter_index)
```

Description

`value = getparam(target_object, parameter_block_name, parameter_name)` returns the value of block parameter `parameter_name` in block `parameter_block_name`.

`value = getparam(target_object, parameter_name)` returns the value of MATLAB variable `parameter_name`.

`value = getparam(target_object, parameter_index)` returns the value of the parameter associated with `parameter_index`.

Examples

Get Block Parameter by Parameter and Block Names

Get the value of block parameter 'Amplitude' of block 'Signal Generator'.

```
tg = slrt;  
getparam(tg, 'Signal Generator', 'Amplitude')
```

```
ans =
```

4

Get MATLAB Variable by Scalar Parameter Name

Get the value of MATLAB variable 'Freq'.

```
tg = slrt;  
getparam(tg, 'Freq')
```

```
ans =
```

20

Get MATLAB Variable by Parameter Structure Name

Get the value of parameter structure 'oscp'.

```
tg = slrt;  
getparam(tg, 'oscp')
```

```
ans =
```

```
G0: 1000000  
G1: 400  
G2: 1000000
```

Get MATLAB Variable by Parameter Structure Field Name

Get the value of MATLAB variable 'oscp.G2'.

```
tg = slrt;  
getparam(tg, 'oscp.G2')
```

```
ans =
```

1000000

Get Block Parameter by Parameter Index

Get the parameter index of block parameter 'Gain' of block 'Gain', and then get its value.

```
tg = slrt;  
pid = getparamid(tg, 'Gain', 'Gain');
```

```
getparam(tg, pid)
ans =
    1000000
```

Get MATLAB Variable by Parameter Index

Get the parameter index of MATLAB variable 'G2', and then get its value.

```
tg = slrt;
pid = getparamid(tg, '', 'G2');
getparam(tg, pid)
ans =
    1000000
```

Input Arguments

target_object — Object representing target computer

SimulinkRealTime.target object

Object that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required Ethernet link settings.

Example: tg

Data Types: struct

parameter_block_name — Hierarchical name of the originating block

character vector

The empty character vector (' ') as a block name marks a MATLAB variable that provides the value for a block parameter. The MATLAB variable is not associated with a particular block.

Example: 'Gain1', ''

parameter_name — Name of the parameter

character vector

The parameter can designate either a block parameter or a MATLAB variable that provides the value for a block parameter. To be accessible via parameter name, the parameter must be observable.

Note: Simulink Real-Time does not support parameters of multiword data types.

Example: 'Gain', 'oscp.G1', 'oscp', 'G2'

parameter_index — Index number of the parameter

nonnegative integer

The parameter index can mark either a block parameter or a MATLAB variable that provides the value for a block parameter. To be accessible via parameter index, the parameter must be observable.

Note: Parameter access by parameter index will be removed in a future release. Access parameters by parameter name instead.

Example: 0, 1

Output Arguments

value — Value of parameter

scalar | complex | structure

Simulink Real-Time does not support parameters of multiword data types.

More About

- “Tunable Block Parameters and MATLAB Variables”
- “Nonobservable Parameters”

See Also

Real-Time Application | Real-Time Application Properties |
SimulinkRealTime.target.getparamid | SimulinkRealTime.target.setparam

Introduced in R2014a

SimulinkRealTime.target.getparamid

Parameter index from parameter hierarchical name

Syntax

```
parameter_index = getparamid(target_object, parameter_block_name,  
parameter_name)  
parameter_index = getparamid(target_object, '', parameter_name)
```

Description

`parameter_index = getparamid(target_object, parameter_block_name, parameter_name)` returns the parameter-list index of a block parameter. The function searches the parameter list by the path to the block and the parameter name.

Enter for `parameter_block_name` the mangled name that the Simulink Coder software uses for code generation. You can determine the mangled name as follows:

- If you do not have special characters in your model, use the `gcb` function.
- If the blocks of interest have special characters, retrieve the mangled name using `tg.showparam = 'on'`.

For example, if carriage return `'\n'` is part of the block path, the mangled name returns with carriage returns replaced by spaces.

Enter the names in full. The names are case-sensitive.

`parameter_index = getparamid(target_object, '', parameter_name)` returns the parameter-list index of a MATLAB variable that provides the value for a block parameter. The function searches the parameter list by the parameter name. The name is case-sensitive.

For the block name argument, enter the empty character vector (`''`).

Examples

Get Block Parameter by Parameter and Block Names

Get the value of block parameter 'Amplitude' of block 'Signal Generator'

```
tg = slrt;
pid = getparamid(tg, 'Signal Generator', 'Amplitude');

getparam(tg, pid)

ans =

     4
```

Get MATLAB Variable by Scalar Parameter Name

Get the value of MATLAB variable 'Freq'

```
tg = slrt;
pid = getparamid(tg, '', 'Freq');

getparam(tg, pid)

ans =

    20
```

Input Arguments

target_object — Object representing target computer

SimulinkRealTime.target object

Object that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required Ethernet link settings.

Example: tg

Data Types: struct

parameter_block_name — Hierarchical name of the originating block

character vector

The empty character vector (' ') as a block name marks a MATLAB variable that provides the value for a block parameter. The MATLAB variable is not associated with a particular block.

Example: 'Gain1', ''

parameter_name — Name of the parameter

character vector

The parameter can designate either a block parameter or a MATLAB variable that provides the value for a block parameter. To be accessible via parameter name, the parameter must be observable.

Note: Simulink Real-Time does not support parameters of multiword data types.

Example: 'Gain', 'osc.p.G1', 'osc.p', 'G2'

Output Arguments

parameter_index — Index number of the parameter

nonnegative integer

The parameter index can mark either a block parameter or a MATLAB variable that provides the value for a block parameter. To be accessible via parameter index, the parameter must be observable.

Note: Parameter access by parameter index will be removed in a future release. Access parameters by parameter name instead.

Example: 0, 1

More About

- “Tunable Block Parameters and MATLAB Variables”
- “Nonobservable Parameters”

See Also

Real-Time Application | Real-Time Application Properties |
SimulinkRealTime.target.getparam | SimulinkRealTime.target.setparam

Introduced in R2014a

SimulinkRealTime.target.getparamname

Block path and parameter name from parameter index

Syntax

```
[block_path, parameter_name] = getparamname(target_object,  
parameter_index)
```

Arguments

target_object	Name of a target object.
parameter_index	Index number of the parameter.

Note: Parameter access by parameter index will be removed in a future release. Access parameters by parameter name instead.

[block_path, parameter_name]	Output vector containing the block path and parameter name for the specified parameter.
---------------------------------	---

Description

[block_path, parameter_name] = getparamname(target_object, parameter_index) returns a vector containing two character vectors (block path and parameter name) for the parameter specified by parameter_index.

Examples

Get the block path and parameter name of parameter index 5:

```
tg = slrt;  
[block_path, parameter_name] = getparamname(tg,5)  
block_path =
```

```
Signal Generator  
parameter_name =  
Amplitude
```

See Also

Real-Time Application | Real-Time Application Properties

Introduced in R2014a

SimulinkRealTime.target.getPCIInfo

Return information about PCI boards installed in target computer

Syntax

```
getPCIInfo(target_object)
getPCIInfo(target_object, 'ethernet')
getPCIInfo(target_object, 'all')
getPCIInfo(target_object, 'verbose')
pci_devices = getPCIInfo(target_object, ___)

getPCIInfo(target_object, 'supported')
pci_devices_supported = getPCIInfo(target_object, 'supported')
```

Description

`getPCIInfo(target_object)` queries the target computer, represented by `target_object`, for installed PCI devices other than Ethernet controllers that the Simulink Real-Time block library supports. To retrieve information about Ethernet controllers, use the `'ethernet'` option.

The software displays in the Command Window information about the PCI devices that `getPCIInfo` found, including:

- PCI bus number
- Slot number
- Assigned IRQ number
- Vendor (manufacturer) name
- Device (board) name
- Device type
- Vendor PCI ID
- Device PCI ID
- Device release version

Before you can use this call, verify that the target computer has started under the Simulink Real-Time kernel and that the Ethernet link is working. The real-time application can be loaded or the loader can be active and waiting for input. You can check these preconditions by verifying that the function `SimulinkRealTime.pingTarget` returns `success`.

Before building the model, you can use `getPCIInfo` to find resources to enter into a driver block dialog box. Such resources include PCI bus number, slot number, and assigned IRQ number.

`getPCIInfo(target_object, 'ethernet')` queries the target computer, represented by `target_object`, for Ethernet controllers that are installed.

`getPCIInfo(target_object, 'all')` displays information about all of the PCI devices found on the target computer represented by `target_object`. This information includes graphics controllers, Ethernet cards, SCSI cards, and devices that are part of the motherboard chip set (for example, PCI-to-PCI bridges).

`getPCIInfo(target_object, 'verbose')` shows the information displayed by `getPCIInfo(target_object, 'all')` for the target computer represented by `target_object`, plus information about the PCI addresses that the BIOS assigns to this board.

`pci_devices = getPCIInfo(target_object, ___)` queries the target computer represented by `target_object` according to the additional arguments you supplied. The call returns a structure containing information about the PCI devices that the software found on the target computer.

`getPCIInfo(target_object, 'supported')` displays a list of the PCI devices supported by the Simulink Real-Time block library. This call does not access the target computer, so the Ethernet link does not have to be active.

`pci_devices_supported = getPCIInfo(target_object, 'supported')` returns a structure containing a list of devices supported by Simulink Real-Time. This call does not access the target computer, so the Ethernet link does not have to be active.

Examples

Display Information for PCI Devices on Default Computer that the Simulink Real-Time Block Library Supports

Start the default target computer with the Simulink Real-Time kernel. Verify the connection between the development and the target computer. At the command prompt, type the command on the development computer.

```
slrtpingtarget
```

```
target_object = slrt;  
getPCIInfo(target_object)
```

```
List of installed PCI devices:
```

```
Measurement Computing    PCI-DIO24  
  Bus 1, Slot 11, IRQ 10  
  DI DO  
  VendorID 0x1307, DeviceID 0x0028,  
    SubVendorID 0x1307, SubDeviceID 0x0028  
  A/D Chan: 0, D/A Chan: 0, DIO Chan: 24  
  Released in: R14SP2 or Earlier  
  
. . .
```

Display Information for Ethernet Controllers on Default Computer

Start the default target computer with the Simulink Real-Time kernel. Verify the connection between the development and target computers. At the MATLAB command prompt, type the command on the development computer.

```
slrtpingtarget
```

```
target_object = slrt;  
getPCIInfo(target_object, 'ethernet')
```

```
List of installed PCI devices:
```

```
Intel                      82541GI_LF  
  Bus 16, Slot 4, IRQ 10  
  Ethernet controller  
  VendorID 0x8086, DeviceID 0x107c, SubVendorID 0x8086,
```

```

    SubDeviceID 0x1376
Released in: R2006b
Notes: Intel Gigabit Ethernet series

```

Display Information for All PCI Devices on Default Computer

Start the default target computer with the Simulink Real-Time kernel. Verify the connection between the development and target computers. At the command prompt, type the command on the development computer.

```
slrtpingtarget
```

```
target_object = slrt;
getPCIInfo(target_object, 'all')
```

```
List of installed PCI devices:
```

```

Intel                               Unknown
  Bus 0, Slot 0, IRQ 0
  Host Bridge
  VendorID 0x8086, DeviceID 0x1130,
    SubVendorID 0x8086, SubDeviceID 0x4532
.
.
.
Measurement Computing      PCI-DIO24
  Bus 1, Slot 11, IRQ 10
  DI DO
  VendorID 0x1307, DeviceID 0x0028,
    SubVendorID 0x1307, SubDeviceID 0x0028
  A/D Chan: 0, D/A Chan: 0, DIO Chan: 24
  Released in: R14SP2 or Earlier
.
.
.

```

Display Verbose Information for All PCI Devices on Default Computer

Start the default target computer with the Simulink Real-Time kernel. Verify the connection between the development and target computers. At the command prompt, type the command on the development computer.

```
slrtpingtarget
```

```
target_object = slrt;
getPCIInfo(target_object, 'verbose')
```

List of installed PCI devices:

```
Intel                               Unknown
  Bus 0, Slot 0, IRQ 0
  Host Bridge
  VendorID 0x8086, DeviceID 0x1130,
    SubVendorID 0x8086, SubDeviceID 0x4532
  BaseClass 6, SubClass 0
  BAR BaseAddress AddressSpace  MemoryType PreFetchable
    0)    E8000000      Memory    32-bit decoder      no
.
.
.
Measurement Computing      PCI-DIO24
  Bus 1, Slot 11, IRQ 10
  DI DO
  VendorID 0x1307, DeviceID 0x0028,
    SubVendorID 0x1307, SubDeviceID 0x0028
  A/D Chan: 0, D/A Chan: 0, DIO Chan: 24
  Released in: R14SP2 or Earlier
  BaseClass FF, SubClass FF
  BAR BaseAddress AddressSpace
    1)      DC00          I/O
    2)      DFF4          I/O
.
.
.
```

Return Information for PCI Devices on Default Computer that the Simulink Real-Time Block Library Supports

Start the default target computer with the Simulink Real-Time kernel. Verify the connection between the development and target computers. At the command prompt, type the command on the development computer. Display the first structure in the vector.

```
slrtpingtarget

target_object = slrt;
pci_devices = getPCIInfo(target_object);
pci_devices(1)

ans =
```

```
    Bus: 1
```



```

        Slot: 11
        VendorID: '1307'
        DeviceID: '28'
        SubVendorID: '1307'
        SubDeviceID: '28'
        BaseClass: 'FF'
        SubClass: 'FF'
        Interrupt: 10
BaseAddresses: [1x6 struct]
        VendorName: 'Measurement Computing'
        Release: 'R14SP2 or Earlier'
        Notes: ''
        DeviceName: 'PCI-DIO24'
        DeviceType: 'DI D0'
        ADChan: '0'
        DAChan: '0'
        DIOChan: '24'

```

Return Information for All PCI Devices on Default Computer

Start the default target computer with the Simulink Real-Time kernel. Verify the connection between the development and target computers. At the command prompt, type the command on the development computer. Display the first structure in the vector.

```

slrtpingtarget

target_object = slrt;
pci_devices = getPCIInfo(target_object, 'all');
pci_devices(1)

ans =

```

```

        Bus: 0
        Slot: 0
        VendorID: '8086'
        DeviceID: '1130'
        SubVendorID: '8086'
        SubDeviceID: '4532'
        BaseClass: '6'
        SubClass: '0'
        Interrupt: 0
BaseAddresses: [1x6 struct]
        VendorName: 'Intel'
        Release: ''

```

```
Notes: ''
DeviceName: 'Unknown'
DeviceType: 'Host Bridge'
ADChan: ''
DACHan: ''
DIOChan: ''
```

Return Verbose Information for All PCI Devices Via `target_object`

Start the default target computer with the Simulink Real-Time kernel. To get the `target_object`, use `SimulinkRealTime.target`. Verify the connection between the development and target computers. At the command prompt, type the command on the development computer. Display the first structure in the vector.

```
SimulinkRealTime.pingTarget('TargetPC1')
```

```
pci_devices = getPCIInfo(target_object, 'verbose');
pci_devices(1)
```

```
ans =
```

```
Bus: 0
Slot: 0
VendorID: '8086'
DeviceID: '1130'
SubVendorID: '8086'
SubDeviceID: '4532'
BaseClass: '6'
SubClass: '0'
Interrupt: 0
BaseAddresses: [1x6 struct]
VendorName: 'Intel'
Release: ''
Notes: ''
DeviceName: 'Unknown'
DeviceType: 'Host Bridge'
ADChan: ''
DACHan: ''
DIOChan: ''
```

Display Information for All PCI Devices that the Simulink Real-Time Block Library Supports

At the command prompt, type the commands on the development computer. The target computer does not have to be active.

```
target_object = SimulinkRealTime.target
getPCIInfo(target_object, 'supported')
```

List of supported PCI devices:

Vendor	Device	Type...
ADLINK	PCI-6208A	AO DI DO...
B&B Electronics (Quatech)	DSCP-200/300 (PXI)	Serial Ports...
.		
.		
.		
Speedgoat	I0321 (PMC-FPGA)	AI (I0321-5)...
Speedgoat	I0331 (PMC-FPGA)	DI DO (LVDS/LVCMOS)...

Return Information for All PCI Devices that the Simulink Real-Time Block Library Supports

At the command prompt, type the commands on the development computer. The target computer does not have to be active.

```
target_object = SimulinkRealTime.target

pci_devices_supported = getPCIInfo(target_object, 'supported');
pci_devices_supported(1)
```

ans =

```

    VendorID: '144A'
    DeviceID: '6208'
SubVendorID: '-1'
SubDeviceID: '-1'
  DeviceName: 'PCI-6208A'
   VendorName: 'ADLINK'
   DeviceType: 'AO DI DO'
         DChan: '8'
         ADChan: '0'
        DIOChan: '4'
         Release: 'R14SP2 or Earlier'
```

Notes: 'PCI-6208A features 8 current outputs w...'

Input Arguments

target_object — Object representing target computer

SimulinkRealTime.target object

Object that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required Ethernet link settings.

Example: `tg`

Data Types: `struct`

Output Arguments

pci_devices — Information about the PCI devices in the target computer

vector

The vector that `getPCIInfo` returns when you call it without an argument contains information only for those PCI devices that the Simulink Real-Time library blocks support.

The vectors returned by `getPCIInfo` with the arguments `'all'` and `'verbose'` contain information about all PCI devices in the target computer. The vectors are identical.

The fields in this structure are:

Bus — PCI bus where device resides

scalar

`Bus` and `Slot` uniquely identify the location of a device or bus adapter in the target computer.

Slot — PCI slot where device resides

scalar

`Slot` and `Bus` uniquely identify the location of a device or bus adapter in the target computer.

VendorID — Identifier for manufacturer of the device

character vector

Hexadecimal numeric character vector containing the identifier that the PCI standards organization assigns to the manufacturer of this device or bus adapter.

DeviceID — Identifier for device among those manufactured by the vendor

character vector

Hexadecimal numeric character vector containing the identifier that the manufacturer assigns to this device or bus adapter.

SubVendorID — Identifier for manufacturer of subsystem

character vector

Hexadecimal numeric character vector containing the identifier that the PCI standards organization assigns to the manufacturer of the entire subsystem (board).

SubDeviceID — Identifier for subsystem among those manufactured by the subvendor

character vector

Hexadecimal numeric character vector containing the identifier that the manufacturer assigns to this subsystem (board).

BaseClass — Standard PCI class of the device

character vector

Hexadecimal numeric character vector containing the standard PCI base classification of this device or bus adapter. **BaseClass** and **SubClass** identify the type and function of the device.

SubClass — Standard PCI subclass of the device

character vector

Hexadecimal numeric character vector containing the standard PCI subclass classification of this device or bus adapter. **SubClass** and **BaseClass** identify the type and function of the device.

Interrupt — IRQ used by the device

scalar

Provides the board-level interrupt that the device or bus adapter uses to trigger I/O with the target computer CPU.

BaseAddresses — Information for each Base Address Register (BAR) used by the device
vector

For each BAR used that this device or bus adapter uses, the vector contains a structure with the following fields:

AddressSpaceIndicator — Indicates whether the address is a memory or I/O address
0 | 1

- 0 — Memory address
- 1 — I/O address

BaseAddress — Memory address used by the device
character vector

Hexadecimal character vector containing the base memory address that the device uses.

MemoryType — Indicates the size of the address decode, 32-bit or 64-bit
0 | 1

Not used if `AddressSpaceIndicator` is 1 (I/O address).

- 0 — 32-bit address decode
- 1 — 64-bit address decode

Prefetchable — Indicates whether the memory is prefetchable
0 | 1

Not used if `AddressSpaceIndicator` is 1 (I/O address).

- 0 — Address is not prefetchable
- 1 — Address is prefetchable

VendorName — Name of vendor of device
character vector

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
character vector

If the Simulink Real-Time block library supports the device, it contains the MATLAB and Simulink release version in which the driver was released. Otherwise, it contains an empty vector.

Notes — Additional information about the device

character vector

Contains additional description of the device or bus adapter.

DeviceName — Name of device

character vector

Identifies the specific device or bus adapter. Set to 'Unknown' for unknown devices or bus adapters.

DeviceType — Identifies the functions of the device

character vector

Contains abbreviations such as 'DI' (digital input) that indicate the function or functions of the device or bus adapter.

ADChan — Number of analog inputs

character vector

Decimal numeric character vector containing the number of analog inputs to the device.

DACHan — Number of analog outputs

character vector

Decimal numeric character vector containing the number of analog outputs from the device.

DIOChan — Number of digital inputs and outputs

character vector

Decimal numeric character vector containing the number of digital inputs and outputs to and from the device.

pci_devices_supported — Information about the PCI devices supported by the product

vector

Vector of information about the devices and bus adapters that the blocks in the Simulink Real-Time block library represent.

The fields are as follows:

VendorID — Identifier for manufacturer of the device

character vector

Hexadecimal numeric character vector containing the identifier that the PCI standards organization assigns to the manufacturer of this device or bus adapter.

DeviceID — Identifier for device among those manufactured by the vendor

character vector

Hexadecimal numeric character vector containing the identifier that the manufacturer assigns to this device or bus adapter.

SubVendorID — Identifier for manufacturer of subsystem

character vector

Hexadecimal numeric character vector containing the identifier that the PCI standards organization assigns to the manufacturer of the entire subsystem (board).

SubDeviceID — Identifier for subsystem among those manufactured by the subvendor

character vector

Hexadecimal numeric character vector containing the identifier that the manufacturer assigns to this subsystem (board).

DeviceName — Name of device

character vector

Identifies the specific device or bus adapter. Set to 'Unknown' for unknown devices or bus adapters.

VendorName — Name of vendor of device

character vector

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

character vector

Contains abbreviations such as 'DI' (digital input) that indicate the function or functions of the device or bus adapter.

DACHan — Number of analog outputs

character vector

Decimal numeric character vector containing the number of analog outputs from the device.

ADChan — Number of analog inputs

character vector

Decimal numeric character vector containing the number of analog inputs to the device.

DIOChan — Number of digital inputs and outputs

character vector

Decimal numeric character vector containing the number of digital inputs and outputs to and from the device.

Releas — MATLAB release version in which driver became available

character vector

If the Simulink Real-Time block library supports the device, it contains the MATLAB and Simulink release version in which the driver was released. Otherwise, it contains an empty vector.

Notes — Additional information about the device

character vector

Contains additional description of the device or bus adapter.

More About

- “PCI Board Information”
- “Command-Line Ethernet Card Selection by Index”
- “PCI Bus I/O Devices”

See Also

Real-Time Application | Real-Time Application Properties

Introduced in R2014a

SimulinkRealTime.target.getscope

Return scope identified by scope number

Syntax

```
scope_object_vector = getscope(target_object)
scope_object = getscope(target_object, scope_number)
scope_object_vector = getscope(target_object, scope_number_vector)
```

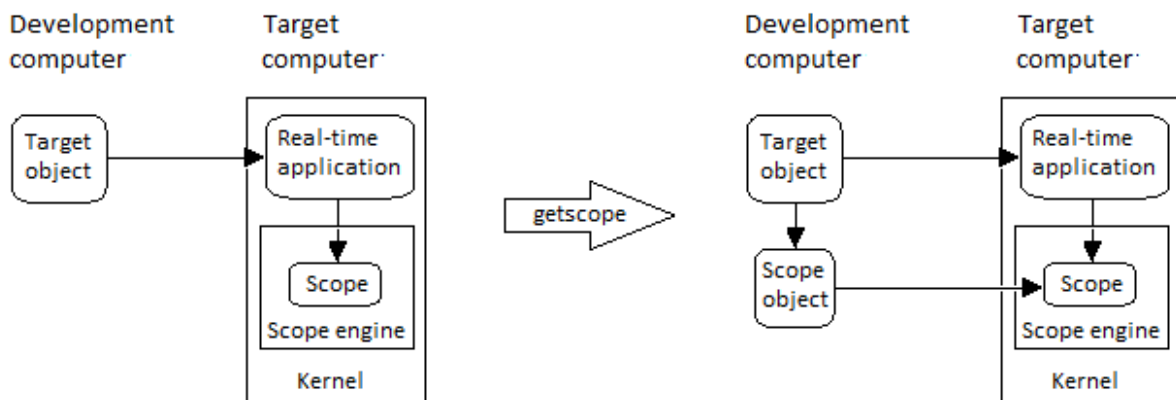
Description

`scope_object_vector = getscope(target_object)` returns a vector containing objects representing all of the existing scopes on the target computer.

`scope_object = getscope(target_object, scope_number)` returns the object representing an existing scope that has the given scope number.

`scope_object_vector = getscope(target_object, scope_number_vector)` returns a vector containing objects representing existing scopes that have the given scope numbers.

If you try to get a nonexistent scope, the result is an error.



Examples

All scopes on the target computer

To view the properties of all scopes on the target, get a vector of scope objects.

Get all scopes on the target computer.

```
tg = slrt;
scope_object_vector = getscope(tg)

scope_object_vector =

Simulink Real-Time Scope
  Application      = xpcosc
  ScopeId          = 1
  Status           = Interrupted
  Type             = Target
  NumSamples       = 500
  NumPrePostSamples = 0
  Decimation       = 1
  TriggerMode      = FreeRun
  TriggerSignal    = 5 : Signal Generator
  TriggerLevel     = 0.000000
  TriggerSlope    = Either
  TriggerScope     = 1
  TriggerSample    = 0
  DisplayMode      = Redraw (Graphical)
  YLimit           = Auto
  Grid             = on
  Signals          = 5 : Signal Generator
                  6 : Sum
```

```
Simulink Real-Time Scope
  Application      = xpcosc
  ScopeId          = 2
  Status           = Interrupted
  Type             = Target
  NumSamples       = 250
  NumPrePostSamples = 0
  Decimation       = 1
  TriggerMode      = FreeRun
  TriggerSignal    = 0 : Gain
  TriggerLevel     = 0.000000
  TriggerSlope    = Either
```

```
TriggerScope      = 2
TriggerSample     = 0
DisplayMode       = Redraw (Graphical)
YLimit            = Auto
Grid              = on
Signals           = 0 : Gain
                  1 : Gain1
                  2 : Gain2
```

```
Simulink Real-Time Scope
Application        = xpcosc
ScopeId           = 3
Status            = Interrupted
Type              = Host
NumSamples        = 250
NumPrePostSamples = 0
Decimation        = 1
TriggerMode       = FreeRun
TriggerSignal     = -1
TriggerLevel      = 0.000000
TriggerSlope     = Either
TriggerScope     = 3
TriggerSample     = 0
StartTime        = -1.000000
Data              = Matrix (250 x 0)
Time              = Matrix (250 x 1)
Signals           = no Signals defined
```

Change the number of samples

To change the number of samples, get a scope object, and then change the scope object property NumSamples.

Get a scope object for scope 1.

```
tg = slrt;
scope_object = getscope(tg,1)
```

```
scope_object =
```

```
Simulink Real-Time Scope
Application        = xpcosc
ScopeId           = 1
Status            = Interrupted
Type              = Target
```

```

NumSamples           = 250
NumPrePostSamples    = 0
Decimation           = 1
TriggerMode          = FreeRun
TriggerSignal        = 5 : Signal Generator
TriggerLevel         = 0.000000
TriggerSlope         = Either
TriggerScope         = 1
TriggerSample        = 0
DisplayMode          = Redraw (Graphical)
YLimit               = Auto
Grid                 = on
Signals              = 5 : Signal Generator
                    6 : Sum

```

Update property NumSamples.

```
scope_object.NumSamples = 500
```

```
scope_object =
```

```
Simulink Real-Time Scope
```

```

Application          = xpcosc
ScopeId              = 1
Status               = Interrupted
Type                 = Target
NumSamples           = 500
NumPrePostSamples    = 0
Decimation           = 1
TriggerMode          = FreeRun
TriggerSignal        = 5 : Signal Generator
TriggerLevel         = 0.000000
TriggerSlope         = Either
TriggerScope         = 1
TriggerSample        = 0
DisplayMode          = Redraw (Graphical)
YLimit               = Auto
Grid                 = on
Signals              = 5 : Signal Generator
                    6 : Sum

```

Vector of scope objects

To view the properties of scopes 1 and 2 on the target computer, get a vector of scope objects.

```
tg = slrt;
scope_object_vector = getscope(tg, [1,2])

scope_object_vector =

Simulink Real-Time Scope
  Application      = xpcosc
  ScopeId         = 1
  Status          = Interrupted
  Type            = Target
  NumSamples      = 500
  NumPrePostSamples = 0
  Decimation      = 1
  TriggerMode     = FreeRun
  TriggerSignal   = 5 : Signal Generator
  TriggerLevel    = 0.000000
  TriggerSlope    = Either
  TriggerScope    = 1
  TriggerSample   = 0
  DisplayMode     = Redraw (Graphical)
  YLimit         = Auto
  Grid            = on
  Signals         = 5 : Signal Generator
                  6 : Sum

Simulink Real-Time Scope
  Application      = xpcosc
  ScopeId         = 2
  Status          = Interrupted
  Type            = Target
  NumSamples      = 250
  NumPrePostSamples = 0
  Decimation      = 1
  TriggerMode     = FreeRun
  TriggerSignal   = 0 : Gain
  TriggerLevel    = 0.000000
  TriggerSlope    = Either
  TriggerScope    = 2
  TriggerSample   = 0
  DisplayMode     = Redraw (Graphical)
  YLimit         = Auto
  Grid            = on
  Signals         = 0 : Gain
                  1 : Gain1
```

2 : Gain2

Input Arguments

target_object — Object representing target computer

`SimulinkRealTime.target` object

Object that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required Ethernet link settings.

Example: `tg`

Data Types: `struct`

scope_number — New scope number

`unsigned integer`

New scope number. This argument is optional. The default value is the next available integer in the target object property `Scopes`.

If you enter the scope number for an existing scope object, the result is an error.

Example: `1`

scope_number_vector — Vector of new scope numbers

`unsigned integer vector`

Vector of new scope numbers. If you enter the scope number for an existing scope object, the result is an error.

Example: `[2, 3]`

Output Arguments

scope_object — Object representing an existing scope

`object`

Object representing an existing scope

scope_object_vector — Vector of objects representing an existing scope

`object`

Vector containing objects representing an existing scope

More About

- “Application and Driver Scripts”

See Also

Real-Time Application | Real-Time File Scope | Real-Time Application Properties | Real-Time Host Scope | Real-Time Target Scope | `SimulinkRealTime.target.addscope` | `SimulinkRealTime.target.remscope`

Introduced in R2014a

SimulinkRealTime.target.getsignal

Value of signal

Syntax

```
signal_value = getsignal(target_object, signal_name)
```

```
signal_value = getsignal(target_object, signal_index)
```

Description

`signal_value = getsignal(target_object, signal_name)` returns the value of signal `signal_name` at the time the request is made. The value is not time-stamped. Successive calls to this function do not necessarily return successive signal values.

`signal_value = getsignal(target_object, signal_index)` returns the value of the signal associated with `signal_index` at the time the request is made. The value is not time-stamped. Successive calls to this function do not necessarily return successive signal values.

Examples

Get Value of Signal by Name

Get the value of signal 'Gain1'.

```
getsignal(tg, 'Gain1')
```

```
ans =
```

```
-3.3869e+006
```

Get Value of Signal by Signal Index

Get the signal index of signal 'Gain1', and then get its value.

```
tg = slrt;
```

```
sid = getsignalid(tg, 'Gain1');  
getsignal(tg, sid)  
ans =  
  
-3.3869e+006
```

Input Arguments

target_object — Object representing target computer

SimulinkRealTime.target object

Object that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required Ethernet link settings.

Example: tg

Data Types: struct

signal_name — Hierarchical name of signal from model

character vector

Simulink Real-Time constructs signal names in two ways:

- For blocks with a single signal, **signal_name** is the same as the block name.
- For blocks with multiple signals, Simulink Real-Time constructs **signal_name** by appending ' /s1', ' /s2', ..., ' /sN' to the block name.

Example: 'Gain2', 'Feedback/Gain1', 'Byte Packing /s2'

signal_index — Index number of the signal

nonnegative integer

To be accessible via signal index, the signal must be observable.

Note: Signal access by signal index will be removed in a future release. Access signals by signal name instead.

Example: 0, 1

Output Arguments

signal_value — Value of signal

number | character vector

Virtual and bus signals, optimized signals, and signals of complex data types are not observable.

More About

- “Nonobservable Signals”

See Also

Real-Time Application | Real-Time Application Properties |
`SimulinkRealTime.target.getsignalid`

Introduced in R2014a

SimulinkRealTime.target.getsignalid

Signal index from signal hierarchical name

Syntax

```
signal_index = getsignalid(target_object, signal_name)
```

Description

`signal_index = getsignalid(target_object, signal_name)` returns the index of a signal from the signal list, based on the path to the block and the signal name.

Enter for `signal_name` the mangled name that the Simulink Coder software uses for code generation. You can determine the mangled name as follows:

- If you do not have special characters in your model, use the `gcb` function.
- If the blocks of interest have special characters, retrieve the mangled name using `tg.showsignals='on'`.

For example, if carriage return `'\n'` is part of the block path, the mangled name returns with carriage returns replaced by spaces.

Enter the names in full. The names are case sensitive.

Examples

Top-Level Block with Single Output

Get signal index for single output of block `Gain1`.

```
tg = slrt;  
getsignalid(tg, 'Gain1')
```

```
ans =
```

6

Lower-Level Block with Single Output

Get signal index for single output of block Feedback/Gain1.

```
tg = slrt;
getsignalid(tg, 'Feedback/Gain1')
```

```
ans =
```

6

Top-Level Block with Multiple Outputs

Get signal index for output signal 2 of block Byte Packing.

```
tg = slrt;
signal_index = getsignalid(tg, 'Byte Packing /s2')
```

```
signal_index =
```

1

Input Arguments

target_object — Object representing target computer

SimulinkRealTime.target object

Object that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required Ethernet link settings.

Example: tg

Data Types: struct

signal_name — Hierarchical name of signal from model

character vector

Simulink Real-Time constructs signal names in two ways:

- For blocks with a single signal, `signal_name` is the same as the block name.

- For blocks with multiple signals, Simulink Real-Time constructs `signal_name` by appending ' /s1', ' /s2', ..., ' /sN' to the block name.

Example: 'Gain2', 'Feedback/Gain1', 'Byte Packing /s2'

Output Arguments

signal_index — Index number of the signal

nonnegative integer

To be accessible via signal index, the signal must be observable.

Note: Signal access by signal index will be removed in a future release. Access signals by signal name instead.

Example: 0, 1

More About

- “Nonobservable Signals”

See Also

Real-Time Application | Real-Time Application Properties |
`SimulinkRealTime.target.getsignal`

Introduced in R2014a

SimulinkRealTime.target.getsignalidsfromlabel

Vector of signal indices

Syntax

```
index_vector = getsignalidsfromlabel(target_object, signal_label)
```

Arguments

<code>target_object</code>	Name of a target object.
<code>signal_label</code>	Signal label (from Simulink model).

Description

```
index_vector = getsignalidsfromlabel(target_object, signal_label)
```

returns a vector of one or more signal indices that are associated with the labeled signal, `signal_label`.

Note: Signal access by signal index will be removed in a future release. Access signals by signal name instead.

You must have labeled the signal for which you request the index using the Simulink **Signal name** parameter. You must have applied a unique label. That is, only one signal has the label `signal_label`.

The Simulink Real-Time software refers to Simulink signal names as signal labels.

Examples

Get the vector of signal indices for a signal labeled `Gain`:

```
tg = slrt;
```

```
getsignalidsfromlabel(tg, 'xpcoscGain')  
ans =
```

```
0
```

More About

- “Signal Properties Controls”

See Also

Real-Time Application | Real-Time Application Properties

Introduced in R2014a

SimulinkRealTime.target.getsignallabel

Signal label for signal index

Syntax

```
signal_label = getsignallabel(target_object, signal_index)
```

Arguments

<code>target_object</code>	Name of a target object.
<code>signal_index</code>	Index number of the signal.

Note: Signal access by signal index will be removed in a future release. Access signals by signal name instead.

Description

`signal_label = getsignallabel(target_object, signal_index)` returns the signal label for the specified signal index, `signal_index`.

You must have labeled the signal for which you request the index using the Simulink **Signal name** parameter. The Simulink Real-Time software refers to Simulink signal names as signal labels.

Examples

Get the signal label for signal index 0:

```
tg = slrt;  
getsignallabel(tg, 0)  
ans =  
  
xpcoscGain
```

More About

- “Signal Properties Controls”

See Also

Real-Time Application | Real-Time Application Properties |
`SimulinkRealTime.target.getsignalidsfromlabel`

Introduced in R2014a

SimulinkRealTime.target.getsignalname

Signal name from index list

Syntax

```
signal_name = getsignalname(target_object, signal_index)
```

Arguments

target_object	Name of a target object.
signal_index	Index number of the signal.

Note: Signal access by signal index will be removed in a future release. Access signals by signal name instead.

signal_name	Output name character vector of the signal.
-------------	---

Description

`signal_name = getsignalname(target_object, signal_index)` returns a character vector from the index list for the specified signal index.

The signal name refers to the block path of the block whose output is the specified signal. The software constructs the name according to the following rules:

- If the block has more than one output port, '/pn' is appended to the signal name, where n is the port number (starting at 1).
- If the output port is not a scalar, '/sn' is appended to the signal name, where n is the index of signal `signal_index` within the vector or matrix. For this purpose, the signals are flattened to one dimension. For example, the signals /s1, /s2, /s3, and /s4 represent a 2 x 2 matrix.

These rules result in the following function behavior for block `Subsystem/path/to/block`:

- If the block has only one output port and the port is a scalar port, the function returns `Subsystem/path/to/block`.
- If the block has one output port, the port is a vector port, and `signal_index` refers to the second element within that vector, the function returns `Subsystem/path/to/block/s2`.
- If the block has three output ports, the second output port outputs a vector, and `signal_index` refers to the seventh element within that vector, the function returns `Subsystem/path/to/block/p2/s7`.
- If the block has three output ports, the second port outputs a scalar, and `signal_index` refers to the output from the second port, the function returns `Subsystem/path/to/block/p2`.

Examples

Get the signal name of signal index 2:

```
tg = slrt;  
sigName = getSignalName(tg,2)  
sigName =  
Gain2
```

See Also

Real-Time Application | Real-Time Application Properties

Introduced in R2014a

SimulinkRealTime.target.load

Download real-time application to target computer

Syntax

```
target_object = load(target_object,real_time_application)
```

Description

`target_object = load(target_object,real_time_application)` loads the application `real_time_application` onto the target computer represented by `target_object`.

The call returns `target_object`, updated with the new state of the target.

Examples

Load xpcosc

Load the real-time application `xpcosc` into target computer `TargetPC1`, represented by target object `tg`. Start the application.

Get the target object.

```
tg = SimulinkRealTime.target('TargetPC1')
```

```
Simulink Real-Time Object
```

```
Connected          = Yes
Application        = loader
```

Load the real-time application.

```
load(tg, 'xpcosc')
```

```
Simulink Real-Time Object
```

```
Connected          = Yes
```

```
Application      = xpcosc
Mode             = Real-Time Single-Tasking
Status          = stopped
CPUOverload     = none

ExecTime        = 0.0000
SessionTime     = 918.5713
StopTime        = 0.200000
SampleTime      = 0.000250
AvgTET          = NaN
MinTET          = 9999999.000000
MaxTET          = 0.000000
ViewMode        = 0

TimeLog          = Vector(0)
StateLog         = Matrix (0 x 2)
OutputLog        = Matrix (0 x 2)
TETLog           = Vector(0)
MaxLogSamples    = 16666
NumLogWraps      = 0
LogMode          = Normal

Scopes           = No Scopes defined
NumSignals       = 7
ShowSignals      = off

NumParameters    = 7
ShowParameters   = off
```

Start the application.

```
start(tg)
```

Input Arguments

target_object — Object representing target computer

SimulinkRealTime.target object

Object that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required Ethernet link settings.

Example: tg

Data Types: struct

real_time_application — Name of real-time application

character vector

Name of the real-time application, without file extension. `real_time_application` can also contain the absolute path to the real-time application, without file extension.

Build the application in the working folder on the development computer. By default, after the Simulink Coder build process is complete, the Simulink Real-Time software calls `SimulinkRealTime.target.load`. If a real-time application was previously loaded, before downloading the new real-time application, `SimulinkRealTime.target.load` unloads the old real-time application.

If you are running the real-time application in Standalone mode, a call to `SimulinkRealTime.target.load` has no effect. To load a new application, rebuild the standalone application files with the new application and transfer the updated files to the target computer using `SimulinkRealTime.fileSystem`. Then, restart the target computer with the new standalone application.

Data Types: char

More About

- “Application and Driver Scripts”

See Also

Real-Time Application | Real-Time Application Properties |
`SimulinkRealTime.target.unload`

Introduced in R2014a

SimulinkRealTime.target.loadparamset

Restore parameter values saved in specified file

Syntax

```
loadparamset(target_object, 'filename')
```

Description

`loadparamset(target_object, 'filename')` restores the real-time application parameter values saved in the file `filename`. Save this file on a local drive of the target computer. You must have a parameter file from a previous run of the `SimulinkRealTime.target.saveparamset` method.

The functions `saveparamset` and `loadparamset` save or load only block parameters. You cannot use these functions to save or load parameters defined only in the model workspace.

Examples

Load Saved Parameters for Model `xpcosc`

Load `xpcosc` parameters from a file named `'xpcosc_params.dat'`

```
tg = slrt;  
loadparamset(tg, 'xpcosc_param.dat')
```

Input Arguments

target_object — Object representing target computer

`SimulinkRealTime.target` object

Object that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required Ethernet link settings.

Example: tg

Data Types: struct

filename — Name of a file in the target computer file system

character vector

In single quotation marks, enter the name of the file that contains the saved parameters.

Example: 'xpcosc_params.dat'

Data Types: char

See Also

Real-Time Application | Real-Time Application Properties |
SimulinkRealTime.target.saveparamset

Introduced in R2014a

SimulinkRealTime.target.ping

Test communication between development and target computers

Syntax

```
status_value = ping(target_object)
```

Description

`status_value = ping(target_object)` tests whether the development computer and the target computer represented by `target_object` can communicate using the settings stored in `target_object`.

Examples

Check communication with default target computer

```
target_object = slrt;  
ping(target_object)
```

```
ans =
```

```
success
```

Check communication with specified target computer, not started

```
target_object = slrt('TargetPC1');  
ping(target_object)
```

```
ans =
```

```
failed
```

Input Arguments

target_object — Object representing target computer
SimulinkRealTime.target object

Object that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required Ethernet link settings.

Example: `tg`

Data Types: `struct`

Output Arguments

status_value — Reports if the kernel is loaded and communication is working

'success' | 'failed'

Simulink Real-Time kernel is loaded and running, and communication is working between the development and target computers.

See Also

Real-Time Application | Real-Time Application Properties

Introduced in R2014a

SimulinkRealTime.target.reboot

Restart target computer

Syntax

```
reboot(target_object)
```

Description

`reboot(target_object)` restarts the target computer. If a target boot disk is still present, `reboot` reloads the Simulink Real-Time kernel.

At the target computer command line, you can use the corresponding command:

```
reboot
```

Examples

Restart Target Computer 'TargetPC1'

Get a target object and restart the target computer that it represents

Get target object for target computer 'TargetPC1'

```
tg = SimulinkRealTime.target('TargetPC1')
```

```
Target: TargetPC1
  Connected           = Yes
  Application         = loader
```

Restart target computer.

```
reboot(tg)
```

Input Arguments

target_object — Object representing target computer
SimulinkRealTime.target object

Object that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required Ethernet link settings.

Example: `tg`

Data Types: `struct`

See Also

“Target Computer Commands” | Real-Time Application | Real-Time Application Properties

Introduced in R2014a

SimulinkRealTime.target.remscope

Remove scope from target computer

Syntax

```
remscope(target_object)
remscope(target_object, scope_number)
remscope(target_object, scope_number_vector)
```

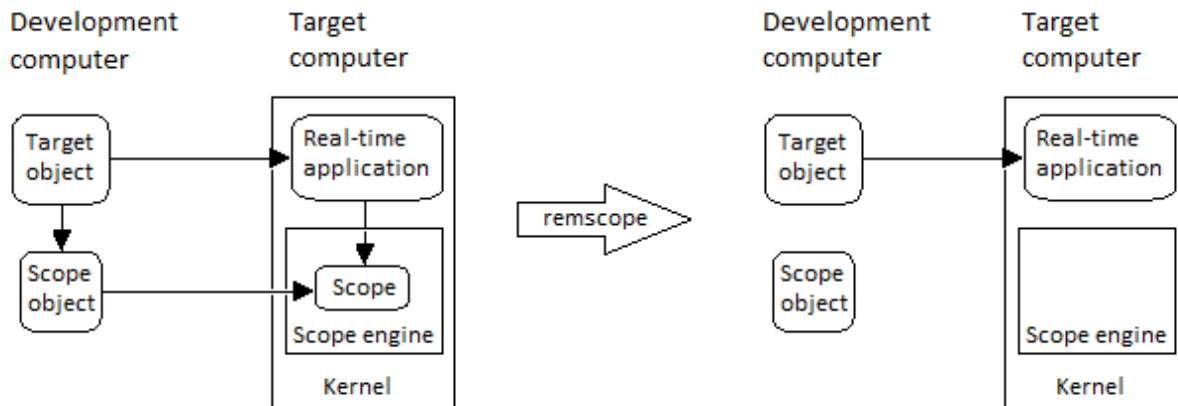
Description

`remscope(target_object)` deletes all scopes from the target computer.

`remscope(target_object, scope_number)` deletes the scope represented by `scope_number` from the target computer.

`remscope(target_object, scope_number_vector)` deletes the scopes represented by the scope numbers listed in `scope_number_vector` from the target computer.

The method `remscope` has no return value. `remscope` does not delete the scope object that represents the scope on the development computer.



You can permanently remove only a scope that is added with the method `addscope`. This scope is outside the model. If you remove a scope that a scope block added inside the model, a subsequent run of that model recreates the scope.

At the target computer command line, you can remove one scope or all scopes:

```
remscope scope_number  
remscope all
```

Examples

Remove All Scopes

```
tg = slrt;  
remscope(tg)
```

Remove one scope

```
tg = slrt;  
remscope(tg,1)
```

Remove vector of two scopes

```
tg = slrt;  
remscope(tg,[1 2])
```

Input Arguments

target_object — Object representing target computer

`SimulinkRealTime.target` object

Object that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required Ethernet link settings.

Example: `tg`

Data Types: `struct`

scope_number — New scope number

`unsigned integer`

New scope number. This argument is optional. The default value is the next available integer in the target object property `Scopes`.

If you enter the scope number for an existing scope object, the result is an error.

Example: 1

scope_number_vector — Vector of new scope numbers

unsigned integer vector

Vector of new scope numbers. If you enter the scope number for an existing scope object, the result is an error.

Example: [2, 3]

See Also

“Target Computer Commands” | Real-Time Application | Real-Time File Scope | Real-Time Application Properties | Real-Time Host Scope | Real-Time Target Scope | `SimulinkRealTime.target.addscope` | `SimulinkRealTime.target.getscope`

Introduced in R2014a

SimulinkRealTime.target.saveparamset

Save real-time application parameter values

Syntax

```
saveparamset(target_object, 'filename')
```

Description

`saveparamset(target_object, 'filename')` saves the real-time application parameter values in the file `filename`. This method saves the file on a local drive of the target computer (C:\ by default). You can later reload these parameters with the `loadparamset` function.

Save real-time application parameter values if you change these parameter values while the application is running in real time. Saving these values enables you to recreate easily real-time application parameter values from several application runs.

The functions `saveparamset` and `loadparamset` save or load only block parameters. You cannot use these functions to save or load parameters defined only in the model workspace.

Examples

Save Parameters for Model `xpcosc`

Save `xpcosc` parameters to a file named `'xpcosc_params.dat'`

```
tg = slrt;  
saveparamset(tg, 'xpcosc_param.dat')
```

Input Arguments

target_object — Object representing target computer
SimulinkRealTime.target object

Object that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required Ethernet link settings.

Example: `tg`

Data Types: `struct`

filename — Name of a file in the target computer file system

character vector

In single quotation marks, enter the name of the file to receive the saved parameters.

Example: `'xpcosc_params.dat'`

Data Types: `char`

See Also

Real-Time Application | Real-Time Application Properties |
`SimulinkRealTime.target.loadparamset`

Introduced in R2014a

SimulinkRealTime.target.setparam

Change value of tunable parameter in real-time application

Syntax

```
setparam(target_object, parameter_block_name, parameter_name,  
parameter_value)  
setparam(target_object, parameter_name, parameter_value)  
  
setparam(target_object, parameter_index, parameter_value)  
setparam(target_object, parameter_index_vec, param_value_cell_array)  
  
history_struct = setparam(target_object, ___)
```

Description

`setparam(target_object, parameter_block_name, parameter_name, parameter_value)` sets the value of a block parameter to a new value. Specify the block parameter by block name and parameter name.

`setparam(target_object, parameter_name, parameter_value)` sets the value of the MATLAB variable to a new value. Specify the variable by parameter name.

`setparam(target_object, parameter_index, parameter_value)` sets the value of the block parameter or MATLAB variable to a new value. Specify the parameter by parameter index.

`setparam(target_object, parameter_index_vec, param_value_cell_array)` sets the value of the target parameter to a new value. Specify the parameter by a vector of parameter indexes and the new value as a cell array.

`history_struct = setparam(target_object, ___)` sets the value of the target parameter to a new value as specified by the parameters. This method returns a structure that stores the parameter specification, previous parameter values, and new parameter values.

Examples

Set Block Parameter by Parameter and Block Names

Set the value of block parameter 'Amplitude' of block 'Signal Generator' to 5.

```
tg = slrt;  
setparam(tg, 'Signal Generator', 'Amplitude', 5)
```

Sweep Block Parameter Values

Sweep the value of block parameter 'Amplitude' of block 'Signal Generator' by steps of 2.

```
tg = slrt;  
for i = 1 : 3  
    setparam(tg, 'Signal Generator', 'Amplitude', (i*2))  
end
```

Set MATLAB Variable by Scalar Parameter Name

Set the value of MATLAB variable 'Freq' to 30.

```
tg = slrt;  
setparam(tg, 'Freq', 30)
```

Set MATLAB Variable by Parameter Structure Field Name

Set the value of MATLAB variable 'oscp.G2' to 10000000.

```
tg = slrt;  
setparam(tg, 'oscp.G2', 10000000)
```

Set Block Parameter by Parameter and Block Names and Return History

Set the value of block parameter 'Amplitude' of block 'Signal Generator' to 5.

```
tg = slrt;  
history_struct = setparam(tg, 'Signal Generator', 'Amplitude', 5)
```

```
history_struct =
```

```
    Source: {'Signal Generator' 'Amplitude'}  
    OldValues: 4
```

```
NewValues: 5
```

Set MATLAB Variable by Scalar Parameter Name and Return History Structure

Set the value of MATLAB variable 'Freq' to 30.

```
tg = slrt;
history_struct = setparam(tg, 'Freq', 30)
```

```
history_struct =
```

```
    Source: {'Freq'}
  OldValues: 20
  NewValues: 30
```

Set MATLAB Variable by Parameter Structure Field Name and Return History Structure

Set the value of MATLAB variable 'oscp.G2' to 10000000.

```
tg = slrt;
history_struct = setparam(tg, 'oscp.G2', 10000000)
```

```
history_struct =
```

```
    Source: {'oscp'}
  OldValues: [1x1 struct]
  NewValues: 10000000
```

Set Block Parameter Value by Parameter Index

Get the signal index of block parameter 'Gain' of block 'Gain1', and then set the parameter value to 10000000.

```
tg = slrt;
pid = getparamid(tg, 'Gain1', 'Gain');
```

```
setparam(tg, pid, 10000000)
```

Set MATLAB Variable Value by Parameter Index

Get the signal index of MATLAB variable 'G2', and then set the parameter value to 10000000.

```
tg = slrt;
pid = getparamid(tg, '', 'G2');
```

```
setparam(tg, pid,10000000)
```

Simultaneously Set Block Parameter Values for Multiple Parameters

Get the signal indexes of block parameters 'Gain' of blocks 'Gain1' and 'Gain2', and then set the parameter values to 10000000 and 400 respectively.

```
tg = slrt;  
pid1 = getparamid(tg, 'Gain1', 'Gain');  
pid2 = getparamid(tg, 'Gain2', 'Gain');  
setparam(tg, [pid1, pid2], {10000000, 400})
```

Input Arguments

target_object — Object representing target computer

SimulinkRealTime.target object

Object that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required Ethernet link settings.

Example: tg

Data Types: struct

parameter_block_name — Hierarchical name of the originating block

character vector

The empty character vector (' ') as a block name marks a MATLAB variable that provides the value for a block parameter. The MATLAB variable is not associated with a particular block.

Example: 'Gain1', ''

parameter_name — Name of the parameter

character vector

The parameter can designate either a block parameter or a MATLAB variable that provides the value for a block parameter. To be accessible via parameter name, the parameter must be observable.

Note: Simulink Real-Time does not support parameters of multiword data types.

Example: 'Gain', 'oscp.G1', 'oscp', 'G2'

parameter_index — Index number of the parameter

nonnegative integer

The parameter index can mark either a block parameter or a MATLAB variable that provides the value for a block parameter. To be accessible via parameter index, the parameter must be observable.

Note: Parameter access by parameter index will be removed in a future release. Access parameters by parameter name instead.

Example: 0, 1

parameter_value — New parameter value

number | character vector | complex | structure

New value with data type as required by parameter.

Example: 1

parameter_index_vec — Vector of parameter index numbers

vector

Parameter indexes returned by `SimulinkRealTime.target.getparamid`

Example: [1,2,3]

param_value_cell_array — New parameter values

cell array

New values with data types as required by parameter. The cell array must contain the same number of values as the parameter index vector.

Example: {1,2,3}

Output Arguments

history_struct — Structure containing changed parameters, old values, and new values

structure

Structure containing the following fields:

- **Source** — Reference to parameters being changed, in the same format as the input argument or arguments. If the input arguments are name character vectors, **Source** contains name character vectors. If the input argument is a parameter index or vector of parameter indexes, **Source** contains a parameter index or a vector of parameter indexes.
- **OldValues** — Values held by parameter or parameters before change.
- **NewValues** — Values held by parameter or parameters before change.

Example:

```
Source: {'Signal Generator' 'Amplitude'}  
OldValues: 4  
NewValues: 5
```

Data Types: struct

More About

- “Tunable Block Parameters and MATLAB Variables”
- “Nonobservable Parameters”

See Also

[Real-Time Application](#) | [Real-Time Application Properties](#) | [SimulinkRealTime.target.getparam](#) | [SimulinkRealTime.target.getparamid](#)

Introduced in R2014a

SimulinkRealTime.target.start

Start execution of real-time application on target computer

Syntax

```
start(target_object)
```

Description

`start(target_object)` starts execution of the real-time application represented by the target object. Before using this method, you must create and load the real-time application on the target computer. If a real-time application is running, this command has no effect.

At the target computer command line, you can use the corresponding command:

```
start
```

Examples

Start real-time application `tg`

Start the real-time application represented by the target object `tg`

```
tg = slrt;  
start(tg)
```

Input Arguments

target_object — Object representing target computer

`SimulinkRealTime.target` object

Object that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required Ethernet link settings.

Example: `tg`

Data Types: `struct`

See Also

“Target Computer Commands” | Real-Time Application | Real-Time Application Properties | `SimulinkRealTime.target.stop`

Introduced in R2014a

SimulinkRealTime.target.stop

Stop execution of real-time application on target computer

Syntax

```
stop(target_object)
```

Description

`stop(target_object)` stops execution of the real-time application represented by the target object. Before using this method, you must create and load the real-time application on the target computer. If a real-time application is not running, this command has no effect.

At the target computer command line, you can use the corresponding command:

```
stop
```

Examples

Stop real-time application `tg`

Stop the real-time application represented by the target object `tg`

```
tg = slrt;  
stop(tg)
```

Input Arguments

target_object — Object representing target computer

SimulinkRealTime.target object

Object that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required Ethernet link settings.

Example: `tg`

Data Types: `struct`

See Also

“Target Computer Commands” | Real-Time Application | Real-Time Application Properties | `SimulinkRealTime.target.start`

Introduced in R2014a

SimulinkRealTime.target.unload

Remove real-time application from target computer

Syntax

```
unload(target_object)
```

Description

`unload(target_object)` removes the loaded real-time application from the target computer. The kernel goes into loader mode and is ready to download new real-time application from the development computer.

If you are running the real-time application in **Stand Alone** mode, this command has no effect. To unload and reload a new standalone real-time application, rebuild the standalone application with the new model. Restart the target computer with the updated standalone application.

Examples

Unload Real-Time Application

Unload the real-time application represented by the target object `tg`.

Unload the real-time application.

```
tg = slrt;  
unload(tg);
```

```
Target: TargetPC1  
  Connected           = Yes  
  Application         = loader
```

Input Arguments

target_object — Object representing target computer
SimulinkRealTime.target object

Object that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required Ethernet link settings.

Example: `tg`

Data Types: `struct`

See Also

Real-Time Application | Real-Time Application Properties |
`SimulinkRealTime.target.load`

Introduced in R2014a

SimulinkRealTime.target.viewTargetScreen

Open real-time window on development computer

Syntax

```
viewTargetScreen(target_object)
```

Description

`viewTargetScreen(target_object)` opens a Simulink Real-Time display window for `target_object`.

The behavior of this function depends on the value for the environment property `TargetScope`:

- **TargetScope** enabled (graphics display) — The function uploads a single image of the target computer screen to the display window. The display is not continually updated because the target computer produces a higher data volume when its graphics card is in VGA mode.

To request a screen update, right-click in the display window and then select **Update Simulink Real-Time Target Screen**.

To save the screen image to a file, right-click in the display window, and then select **Save as image**.

- **TargetScope** disabled (text display) — The function transfers text output once every second to the development computer and displays it in the window.

To save the text output to a file, right-click in the display window, and then select **Save as text file**.

Examples

View Screen for Default Target Computer

Get target object for default computer, open window display with target computer screen

```
tg = slrt;  
viewTargetScreen(tg)
```

View Screen for Target Computer 'TargetPC1'

Get target object for 'TargetPC1', open window display with target computer screen

```
tg = slrt('TargetPC1');  
viewTargetScreen(tg)
```

Input Arguments

target_object — Object representing target computer

SimulinkRealTime.target object

Object that represents the target computer. Before calling this function, make sure that you start the target computer with the Simulink Real-Time kernel and apply the required Ethernet link settings.

Example: tg

Data Types: struct

See Also

Real-Time Application Properties | Real-Time Application

Introduced in R2014a

Real-Time File Scope

Record time-domain data on target computer file system

Description

Controls and accesses properties of file scopes.

The scope gets a data package from the kernel and stores the data in a file on the target computer file system. Depending on the setting of `WriteMode`, the file size is or is not continuously updated. You can transfer the data to another computer for examination or plotting.

The `NumSamples` parameter works with the `autorestart` setting.

- `Autorestart` is on — When the scope triggers, the scope collects data into a memory buffer. A background task examines the buffer and writes data to the disk continuously, appending new data to the end of the file. When the scope reaches the number of samples that you specified, it starts collecting data again, overwriting the memory buffer. If the background task cannot keep pace with data collection, data can be lost.
- `Autorestart` is off — When the scope triggers, the scope collects data into a memory buffer up to the number of samples that you specified, and then stops. A background task examines the buffer and writes data to the disk continuously, appending the new data to the end of the file.

The following limitations exist:

- You can have a maximum of eight files open on the target computer at the same time.
- The largest single file that you can create is 4 GB.
- A fully qualified folder name can have a maximum of 248 characters, including the drive letter, colon, and backslash.
- A fully qualified file name can have a maximum of 260 characters. The file part can have 12 characters—a maximum of eight characters for the file name, one character for the period, and a maximum of three characters for the file extension. If the file name is longer than eight characters, the software truncates it to six characters and adds '~1' to the end of the file name.

The following lexical rules exist:

- Function names are case sensitive. Type the entire name.
- Property names are not case sensitive. You do not need to type the entire name, as long as the characters that you type are unique for the property.

You can invoke some of the scope object properties and functions from the target computer command line when you have loaded the real-time application.

Create Object

`SimulinkRealTime.target.addscope`

Properties

Use scope object properties to select signals that you want to acquire, set triggering modes, and access signal information from the real-time application.

To get the value of a readable scope object property from a scope object:

```
scope_object = getscope(target_object, scope_number);  
value = scope_object.scope_object_property
```

To get the Decimation of scope 3:

```
scope_object = getscope(tg, 3);  
value = scope_object.Decimation
```

To set the value of a writable scope property from a scope object:

```
scope_object = getscope(target_object, scope_number);  
scope_object.scope_object_property = new_value
```

To set the Decimation of scope 3:

```
scope_object = getscope(tg, 3);  
scope_object.Decimation = 10
```

Not all properties are user-writable. For example, after you create the scope, property Type is not writable.

File Scope Properties

AutoRestart — Restart acquisition after acquiring required number of samples

'off' (default) | 'on'

Possible values:

- 'on' — The scope collects data up to `NumSamples`, and then starts over again, appending the new data to the end of the signal data file.
- 'off' — The scope collects data up to `NumSamples`, and then stops.

If the named signal data file exists when you start the real-time application, the software overwrites the old data with the new signal data.

To use the `DynamicFileName` property, set `AutoRestart` to 'on'.

DynamicFileName — Create file names for multiple log files

'off' (default) | 'on'

Enables the file scope to create multiple log files dynamically.

To use the `DynamicFileName` property, set `AutoRestart` to 'on'.

Configure `Filename` to create incrementally numbered file names for the multiple log files. If you do not configure `Filename` as required, the software generates an error when you try to start the scope.

You can enable the creation of up to 99999999 files (<%%%%%%>.dat). The length of a file name, including the specifier, cannot exceed eight characters.

Filename — File name for signal data

'C:\data.dat' (default) | character vector

Provide a name for the file that contains the signal data. For file scopes that you create through the MATLAB interface, no name is initially assigned to `FileName`. After you start the scope, the software assigns a name for the file that is 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 increment dynamically. Use a base file name, an underscore (`_`), and a `< >` specifier. Within the specifier, enter one to eight `%` symbols. Each symbol `%` represents a decimal location in the file name. The specifier can appear anywhere in the file name. For example, the following value for `Filename`, `C:\work\file_<%%>.dat` creates file names with the following pattern:

```
file_001.dat
```

```
file_002.dat  
file_003.dat
```

The last file name of this series is `file_999.dat`. If the block is still logging data when the last file reaches its maximum size, the function restarts and overwrites the first file in the series. If you do not retrieve the data from existing files before they are overwritten, the data is lost.

MaxWriteFileSize — Maximum size of signal data file, in bytes

536870912 (default) | unsigned integer

Provide the maximum size of `Filename`, in bytes. This value must be a multiple of `WriteSize`.

When the size of a log file reaches `MaxWriteFileSize`, the software increments the number in the file name and logs data to the new file. The software logs data to successive files until it fills the file with the highest file number that you specified. If the software cannot create additional log files, it overwrites the first log file.

WriteMode — File allocation table update policy

'Lazy' (default) | 'Commit'

Specify when a file allocation table (FAT) entry is updated. Both 'Lazy' and 'Commit' modes write the signal data to the file. With 'Commit' mode, each file write operation simultaneously updates the FAT entry for the file.

'Commit' mode is slower than 'Lazy' mode. 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. If the system stops responding before the file is closed, the file system does not necessarily know the actual file size. The file contents are intact, but not easily accessible.

WriteSize — Block size, in bytes, of output data

512 (default) | unsigned integer

Enter the block size, in bytes, of the data chunks. This parameter specifies that a memory buffer, of length `NumSamples`, collects data in multiples of `WriteSize`. Using a block size that is the same as the disk sector size provides better performance.

If your system stops responding, you can expect to lose an amount of data equal to the size of `WriteSize`.

Common Scope Properties

Application — Name of the real-time application associated with this scope object

character vector

Read-only name of the real-time application associated with this scope object.

Decimation — Samples to acquire

1 (default) | unsigned integer

Scope acquires every Decimationth sample.

NumPrePostSamples — Samples collected before or after a trigger event

0 (default) | integer

Number of samples collected before or after a trigger event. 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.

NumSamples — Number of contiguous samples captured

unsigned integer

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

The scope writes data samples into a memory buffer of size `NumSamples`. If the scope stops before capturing this number of samples, the scope writes zeroes after the collected data to the end of the buffer. Know what type of data you are collecting, because it is possible that your data contains zeroes.

ScopeId — Unique numeric index

unsigned integer

Read-only numeric index, unique for each scope.

Signals — Signal indexes to display on scope

unsigned integer vector

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

Status — State of scope acquisition

'Acquiring' | 'Ready for being Triggered' | 'Interrupted' | 'Finished'

Read-only state value:

- 'Acquiring' — The scope is acquiring data.
- 'Ready for being Triggered' — The scope is waiting for a trigger.
- 'Interrupted' — The scope is not running (interrupted).
- 'Finished' — The scope has finished acquiring data.

TriggerLevel — Signal trigger crossing value

numeric

If `TriggerMode` is 'Signal', this parameter indicates the value that 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.

TriggerMode — Scope trigger mode

'FreeRun' (default) | 'software' | 'signal' | 'scope'

Trigger mode for a scope:

- 'freerun' — The scope triggers on every sample time.
- 'software' — The scope triggers from the Command Window.
- 'signal' — The scope triggers when a designated signal changes state.
- 'scope' — The scope triggers when a designated scope triggers.

TriggerSample — Trigger sample for scope trigger

0 (default) | -1 | integer

If `TriggerMode` is 'Scope', then `TriggerSample` specifies on which sample of the triggering scope the current scope triggers.

For example, if `TriggerSample` is 0 (default), the current scope triggers on sample 0 (first sample acquired) of the triggering scope. In this case, the two scopes are synchronized with each other.

If `TriggerSample` is 1, the current scope triggers on sample 1 (second sample acquired) of the triggering scope. In this case, the two scopes have a one-sample offset.

Setting `TriggerSample` to -1 means that the current scope triggers at the end of the acquisition cycle of the triggering scope. In this case, the triggered scope acquires its first sample one sample after the last sample of the triggering scope.

TriggerScope — Scope for scope trigger

unsigned integer

If `TriggerMode` is `'Scope'`, this parameter identifies the scope to use for a trigger. To trigger a scope when another scope is triggered, set the slave scope property `TriggerScope` to the scope index of the master scope.

TriggerSignal — Signal for signal trigger

unsigned integer

If `TriggerMode` is `'Signal'`, this parameter identifies the block output signal to use for triggering the scope. Identify the signal with a signal index from the target object property `Signal`.

TriggerSlope — Trigger slope for signal trigger

'Either' (default) | 'Rising' | 'Falling'

If `TriggerMode` is `'Signal'`, `TriggerSlope` indicates the signal behavior that triggers the scope.

- `'Either'` — The signal triggers the scope when it crosses `triggerlevel` in either the rising or falling directions.
- `'Rising'` — The signal triggers the scope when it crosses `triggerlevel` in the rising direction.
- `'Falling'` — The signal triggers the scope when it crosses `triggerlevel` in the falling direction.

Type — Type of scope

'Host' (default) | 'Target' | 'File'

Read-only property that determines how the scope collects and displays its data:

- `'Host'` — The scope collects data on the target computer and displays it on the development computer.
- `'Target'` — The scope collects data on the target computer and displays it on the target computer monitor.
- `'File'` — The scope collects and stores data on the target computer.

Object Functions

`SimulinkRealTime.fileScope.addsignal`

Add signals to file scope represented by scope object

`SimulinkRealTime.fileScope.remsignal`

Remove signals from file scope represented by scope object

<code>SimulinkRealTime.fileScope.start</code>	Start execution of file scope on target computer
<code>SimulinkRealTime.fileScope.stop</code>	Stop execution of file scope on target computer
<code>SimulinkRealTime.fileScope.trigger</code>	Software-trigger start of data acquisition for file scope

Examples

Build and Run Real-Time Application with File Scope

Build and download `xpcosc` and execute the real-time application with a file scope.

Open, build, and download the real-time application.

```
ex_model = 'xpcosc';  
open_system(ex_model);  
rtwbuild(ex_model);  
tg = SimulinkRealTime.target
```

```
Target: TargetPC1  
  Connected           = Yes  
  Application         = xpcosc  
  Mode                = Real-Time Single-Tasking  
  Status              = stopped  
  CPUOverload        = none  
  
  ExecTime            = 0.0000  
  SessionTime        = 601.8748  
  StopTime            = 0.200000  
  SampleTime         = 0.000250  
  AvgTET              = NaN  
  MinTET              = Inf  
  MaxTET              = 0.000000  
  ViewMode           = 0  
  
  TimeLog             = Vector(0)  
  StateLog            = Matrix (0 x 2)  
  OutputLog           = Matrix (0 x 2)  
  TETLog              = Vector(0)  
  MaxLogSamples       = 16666  
  NumLogWraps         = 0  
  LogMode             = Normal
```



```
Scopes           = No Scopes defined
NumSignals       = 7
ShowSignals      = off

NumParameters    = 7
ShowParameters   = off
```

Add and configure file scope 1.

```
sc1 = addscope(tg, 'file', 1);
addsignal(sc1, 4);
addsignal(sc1, 5)
```

```
ans =
```

```
Simulink Real-Time Scope
```

```
Application      = xpcosc
ScopeId          = 1
Status           = Interrupted
Type             = File
NumSamples       = 250
NumPrePostSamples = 0
Decimation       = 1
TriggerMode      = FreeRun
TriggerSignal    = 4 : Integrator1
TriggerLevel     = 0.000000
TriggerSlope     = Either
TriggerScope     = 1
TriggerSample    = 0
FileName         = unset
WriteMode        = Lazy
WriteSize        = 512
AutoRestart      = off
DynamicFileName  = off
MaxWriteFileSize = 536870912
Signals          = 4 : Integrator1
                  5 : Signal Generator
```

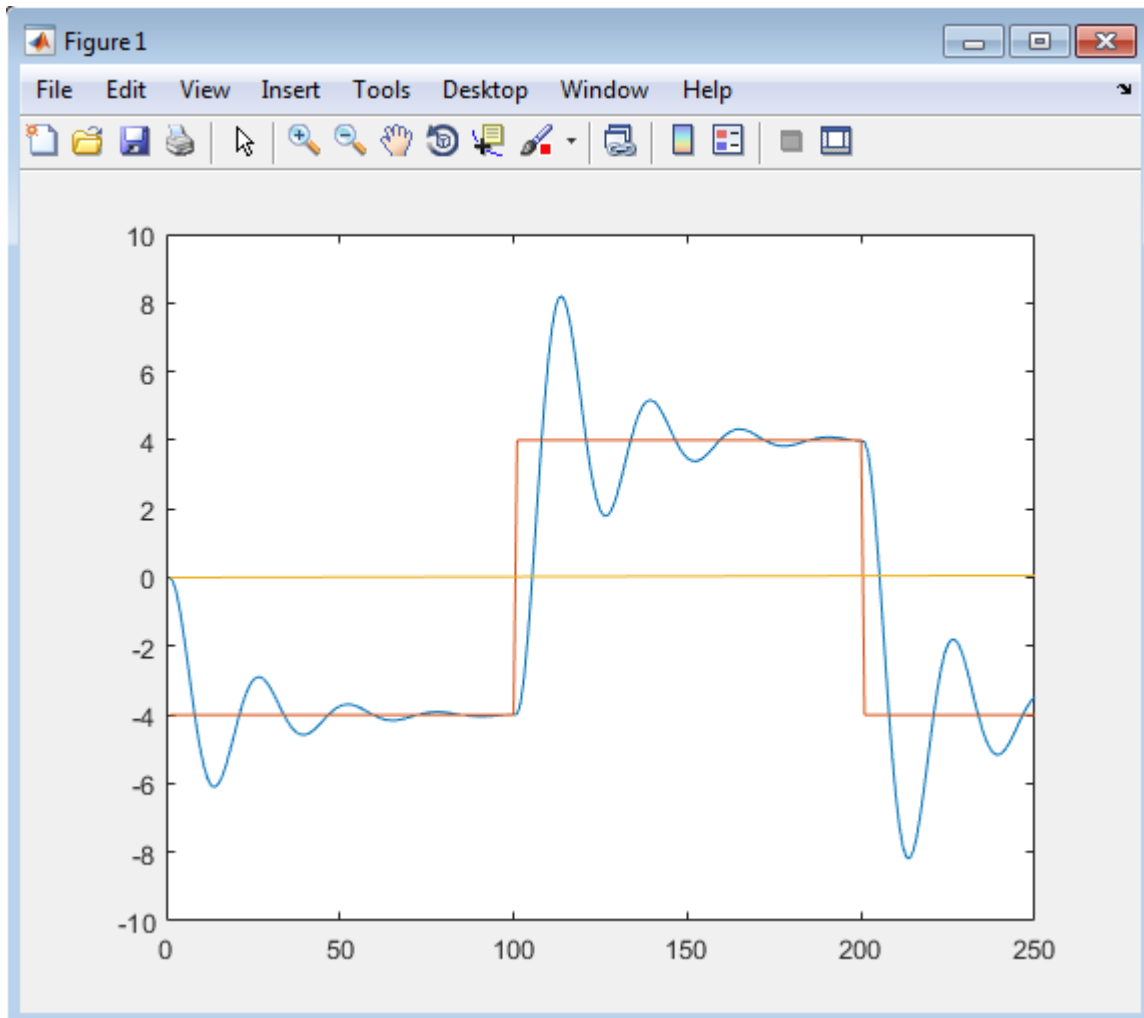
Run the real-time application for 10 seconds.

```
tg.StopTime = 10;
start(sc1);
start(tg);
pause(10);
```

```
stop(tg);  
stop(sc1);
```

Download and display the file scope data.

```
fsys = SimulinkRealTime.fileSystem(tg);  
fh = fopen(fsys, sc1.FileName);  
data = fread(fsys, fh);  
uint8_data = uint8(data);  
plottable_data = ...  
    SimulinkRealTime.utils.getFileScopeData(uint8_data);  
plot(plottable_data.data)
```



Unload the real-time application.

```
unload(tg)
```

```
Target: TargetPC1  
Connected = Yes
```

Application = loader

See Also

“Target Computer Commands” | Real-Time Host Scope | Real-Time Application | Real-Time Application Properties | Real-Time Target Scope | SimulinkRealTime.target.getscope | SimulinkRealTime.target.remscope

More About

- “Data Logging With a File Scope”
- “Simulink Real-Time Scope Usage”
- “File Scope Usage”

Introduced in R2014a

SimulinkRealTime.fileScope.addsignal

Add signals to file scope represented by scope object

Syntax

```
addsignal(scope_object_vector, signal_index_vector)
```

Arguments

<code>scope_object_vector</code>	Name of a single scope object or the name of a vector of scope objects.
<code>signal_index_vector</code>	For one signal, use a single number. For two or more signals, enclose numbers in brackets and separate with commas.

Description

`addsignal(scope_object_vector, signal_index_vector)` adds signals to a scope object. Specify the signals by their indices, which you can retrieve using the target object method `SimulinkRealTime.target.getsignalid`. If `scope_object_vector` has two or more scope objects, the same signals are assigned to each scope.

Before you can add a signal to a scope, you must stop the scope.

At the target computer command line, you can add multiple signals to the scope:

```
addsignal scope_index = signal_index, signal_index, . . .
```

Examples

The following examples use model `xpcosc`.

Add signals `Integrator1` and `Signal Generator` to scope object `sc1`.

```
tg = slrt;  
sc1 = addscope(tg, 'file', 1);  
s0 = getsignalid(tg, 'Signal Generator');  
s1 = getsignalid(tg, 'Integrator1');  
addsignal(sc1, [s0, s1]);
```

The scope object property **Signals** is updated to include the added signals. Type **sc1** to display the properties and values for scope **sc1**.

More About

- “Find Signal and Parameter Indexes”
- “File Scope Usage”

See Also

“Target Computer Commands” | Real-Time File Scope | Real-Time Application | Real-Time Application Properties | `SimulinkRealTime.fileScope.remsignal`

Introduced in R2014a

SimulinkRealTime.fileScope.remsignal

Remove signals from file scope represented by scope object

Syntax

```
remsignal(scope_object)  
remsignal(scope_object, signal_index_vector)
```

Arguments

<code>scope_object_vector</code>	Scope object or vector of scope objects. The target object methods <code>addscope</code> or <code>getscope</code> create scope objects.
<code>signal_index_vector</code>	Index numbers from the scope object property <code>Signals</code> . This argument is optional. If it is left out, all signals are removed.

Description

`remsignal(scope_object)` removes all signals from a scope object.

`remsignal(scope_object, signal_index_vector)` removes signals from a scope object. Specify the signals by their indices, which you can retrieve using the target object method `getsignalid`. If `scope_object` is a vector containing two or more scope objects, the same signals are removed from each scope.

Before you can remove a signal from a scope, you must stop the scope.

At the target computer command line, you can remove multiple signals from the scope:

```
remsignal scope_index = signal_index, signal_index, . . .
```

`signal_index` is optional. If it is left out, all signals are removed.

Examples

The following examples use model `xpcosc`.

Remove all signals from the scope represented by the scope object `sc1`:

```
tg = slrt;  
sc1 = getscope(tg,1);  
remsignal(sc1)
```

Remove signals `Integrator1` and `Signal Generator` from the scope on the target computer:

```
tg = slrt;  
sc1 = getscope(tg,1);  
s0 = getsignalid(tg,'Signal Generator');  
s1 = getsignalid(tg,'Integrator1');  
remsignal(sc1,[s0,s1])
```

More About

- “Find Signal and Parameter Indexes”
- “File Scope Usage”

See Also

“Target Computer Commands” | Real-Time File Scope | Real-Time Application | Real-Time Application Properties | `SimulinkRealTime.fileScope.addsignal`

Introduced in R2014a

SimulinkRealTime.fileScope.start

Start execution of file scope on target computer

Syntax

```
start(scope_object)
start(scope_object_vector)
start([scope_object1, scope_object2, . . . , scope_objectN])
```

Arguments

<code>scope_object</code>	Name of single vector object.
<code>scope_object_vector</code>	Name of vector of scope objects.

Description

`start(scope_object)` starts a scope on the target computer represented by a scope object on the development computer. Whether data acquisition starts depends on the trigger settings.

Before using this method, you must create a scope. To create a scope, use the target object method `addscope` or add Simulink Real-Time scope blocks to your Simulink model.

Alternative syntaxes are `start(scope_object_vector)` and `start([scope_object1, scope_object2, . . . , scope_objectN])`.

At the target computer command line, you can use the corresponding command:

```
startscope scope_index
startscope all
```

Examples

Start one scope with the scope object `sc1`:

```
tg = slrt;  
sc1 = getscope(tg,1)  
start(sc1)
```

Start two scopes, 1 and 2:

```
tg = slrt;  
somescope = getscope(tg,[1,2])  
start(somescope)
```

or

```
tg = slrt;  
sc1 = getscope(tg,1)  
sc2 = getscope(tg,2)  
start([sc1,sc2])
```

Start all scopes:

```
tg = slrt;  
allscopes = getscope(tg)  
start(allscopes)
```

More About

- “Trace Signals at Target Computer Command Line”
- “File Scope Usage”

See Also

“Target Computer Commands” | Real-Time File Scope | Real-Time Application | Real-Time Application Properties | `SimulinkRealTime.fileScope.stop`

Introduced in R2014a

SimulinkRealTime.fileScope.stop

Stop execution of file scope on target computer

Syntax

```
stop(scope_object)
stop(scope_object_vector)
stop([scope_object1, scope_object2, . . . , scope_objectN])
```

Arguments

<code>scope_object</code>	Name of single vector object.
<code>scope_object_vector</code>	Name of vector of scope objects.

Description

`stop(scope_object)` stops a scope on the target computer represented by a scope object on the development computer.

Alternative syntaxes are `stop(scope_object_vector)` and `stop([scope_object1, scope_object2, . . . , scope_objectN])`.

At the target computer command line, you can use the corresponding command:

```
stopscope scope_index
stopscope all
```

Examples

Stop one scope with the scope object `sc1`:

```
tg = slrt;
sc1 = getscope(tg,1)
stop(sc1)
```

Stop two scopes, 1 and 2:

```
tg = slrt;  
somescope = getscope(tg,[1,2])  
stop(somescope)
```

or

```
tg = slrt;  
sc1 = getscope(tg,1)  
sc2 = getscope(tg,2)  
stop([sc1,sc2])
```

Stop all scopes:

```
tg = slrt;  
allscopes = getscope(tg)  
stop(allscopes)
```

More About

- “Trace Signals at Target Computer Command Line”
- “File Scope Usage”

See Also

“Target Computer Commands” | Real-Time File Scope | Real-Time Application | Real-Time Application Properties | `SimulinkRealTime.fileScope.start`

Introduced in R2014a

SimulinkRealTime.fileScope.trigger

Software-trigger start of data acquisition for file scope

Syntax

```
trigger(scope_object_vector)
```

Arguments

<code>scope_object_vector</code>	Name of a single scope object, name of a vector of scope objects, or a list of scope object names in a vector form [<code>scope_object1</code> , <code>scope_object2</code>].
----------------------------------	---

Description

`trigger(scope_object_vector)` triggers the scope represented by the scope object to acquire the number of data points in the scope object property `NumSamples`.

If the scope object property `TriggerMode` has a value of `'Software'`, this function is the only way to trigger the scope. However, this function can be used on any scope, regardless of trigger mode setting. For example, if a scope did not trigger because the triggering criteria were not met, this function can be used to force the scope to trigger.

Examples

Using model `xpcosc`, set a single file scope to software trigger, trigger the acquisition of one set of samples, read the file, and plot the data on the host.

```
tg = slrt;  
tg.StopTime = Inf  
sc1 = addscope(tg,'file',1);  
sc1.FileName = 'data.dat';  
addsignal(sc1, 4)  
sc1.TriggerMode = 'software';
```

```
start(tg)
start(sc1)
trigger(sc1)
pause(0.5)
stop(sc1)
stop(tg)
fsys = SimulinkRealTime.fileSystem(tg);
hdl = fopen(fsys,'data.dat');
ddata = fread(fsys,hdl);
fclose(fsys, hdl);
mdata = SimulinkRealTime.utils.getFileScopeData(ddata);
plot(mdata.data(:,2),mdata.data(:,1))
```

More About

- “File Scope Usage”

See Also

“Target Computer Commands” | Real-Time File Scope | Real-Time Application | Real-Time Application Properties

Introduced in R2014a

Real-Time Host Scope

Display time-domain data on development computer screen

Description

Controls and accesses properties of host scopes.

The kernel acquires a data package and sends it to the scope on the target computer. The scope waits for an upload command from the development computer, and then uploads the data. The development computer displays the data by using Simulink Real-Time Explorer or other MATLAB functions.

The following rules exist:

- Function names are case sensitive. Type the entire name.
- Property names are not case sensitive. You do not need to type the entire name, as long as the characters that you type are unique for the property.

You can invoke some of the scope object properties and functions from the target computer command line when you have loaded the real-time application.

Create Object

`SimulinkRealTime.target.addscope`

Properties

Use scope object properties to select signals that you want to acquire, set triggering modes, and access signal information from the real-time application.

To get the value of a readable scope object property from a scope object:

```
scope_object = getscope(target_object, scope_number);  
value = scope_object.scope_object_property
```

To get the Decimation of scope 3:

```
scope_object = getscope(tg, 3);  
value = scope_object.Decimation
```

To set the value of a writable scope property from a scope object:

```
scope_object = getscope(target_object, scope_number);  
scope_object.scope_object_property = new_value
```

To set the Decimation of scope 3:

```
scope_object = getscope(tg, 3);  
scope_object.Decimation = 10
```

Not all properties are user-writable. For example, after you create the scope, property Type is not writable.

Host Scope Properties

Data — Signal data from host scope

matrix

Contains read-only output data for a single data package from a scope.

Time — Time data from host scope

vector

Contains read-only time data for a single data package from a scope.

Common Scope Properties

Application — Name of the real-time application associated with this scope object

character vector

Read-only name of the real-time application associated with this scope object.

Decimation — Samples to acquire

1 (default) | unsigned integer

Scope acquires every Decimationth sample.

NumPrePostSamples — Samples collected before or after a trigger event

0 (default) | integer

Number of samples collected before or after a trigger event. 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.

NumSamples — Number of contiguous samples captured

unsigned integer

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

The scope writes data samples into a memory buffer of size **NumSamples**. If the scope stops before capturing this number of samples, the scope writes zeroes after the collected data to the end of the buffer. Know what type of data you are collecting, because it is possible that your data contains zeroes.

ScopeId — Unique numeric index

unsigned integer

Read-only numeric index, unique for each scope.

Signals — Signal indexes to display on scope

unsigned integer vector

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

Status — State of scope acquisition

'Acquiring' | 'Ready for being Triggered' | 'Interrupted' | 'Finished'

Read-only state value:

- 'Acquiring' — The scope is acquiring data.
- 'Ready for being Triggered' — The scope is waiting for a trigger.
- 'Interrupted' — The scope is not running (interrupted).
- 'Finished' — The scope has finished acquiring data.

TriggerLevel — Signal trigger crossing value

numeric

If **TriggerMode** is 'Signal', this parameter indicates the value that 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.

TriggerMode — Scope trigger mode

'FreeRun' (default) | 'software' | 'signal' | 'scope'

Trigger mode for a scope:

- 'freerun' — The scope triggers on every sample time.

- `'software'` — The scope triggers from the Command Window.
- `'signal'` — The scope triggers when a designated signal changes state.
- `'scope'` — The scope triggers when a designated scope triggers.

TriggerSample — Trigger sample for scope trigger

0 (default) | -1 | integer

If `TriggerMode` is `'Scope'`, then `TriggerSample` specifies on which sample of the triggering scope the current scope triggers.

For example, if `TriggerSample` is 0 (default), the current scope triggers on sample 0 (first sample acquired) of the triggering scope. In this case, the two scopes are synchronized with each other.

If `TriggerSample` is 1, the current scope triggers on sample 1 (second sample acquired) of the triggering scope. In this case, the two scopes have a one-sample offset.

Setting `TriggerSample` to -1 means that the current scope triggers at the end of the acquisition cycle of the triggering scope. In this case, the triggered scope acquires its first sample one sample after the last sample of the triggering scope.

TriggerScope — Scope for scope trigger

unsigned integer

If `TriggerMode` is `'Scope'`, this parameter identifies the scope to use for a trigger. To trigger a scope when another scope is triggered, set the slave scope property `TriggerScope` to the scope index of the master scope.

TriggerSignal — Signal for signal trigger

unsigned integer

If `TriggerMode` is `'Signal'`, this parameter identifies the block output signal to use for triggering the scope. Identify the signal with a signal index from the target object property `Signal`.

TriggerSlope — Trigger slope for signal trigger

'Either' (default) | 'Rising' | 'Falling'

If `TriggerMode` is `'Signal'`, `TriggerSlope` indicates the signal behavior that triggers the scope.

- `'Either'` — The signal triggers the scope when it crosses `triggerlevel` in either the rising or falling directions.

- 'Rising' — The signal triggers the scope when it crosses `triggerlevel` in the rising direction.
- 'Falling' — The signal triggers the scope when it crosses `triggerlevel` in the falling direction.

Type — Type of scope

'Host' (default) | 'Target' | 'File'

Read-only property that determines how the scope collects and displays its data:

- 'Host' — The scope collects data on the target computer and displays it on the development computer.
- 'Target' — The scope collects data on the target computer and displays it on the target computer monitor.
- 'File' — The scope collects and stores data on the target computer.

Object Functions

<code>SimulinkRealTime.hostScope.addsignal</code>	Add signals to host scope represented by scope object
<code>SimulinkRealTime.hostScope.remsignal</code>	Remove signals from host scope represented by scope object
<code>SimulinkRealTime.hostScope.start</code>	Start execution of host scope on target computer
<code>SimulinkRealTime.hostScope.stop</code>	Stop execution of host scope on target computer
<code>SimulinkRealTime.hostScope.trigger</code>	Software-trigger start of data acquisition for host scope

Examples

Build and Run Real-Time Application with Host Scope

Build and download `xpcosc` and execute the real-time application with a host scope.

Open, build, and download the real-time application.

```
ex_model = 'xpcosc';
open_system(ex_model);
rtwbuild(ex_model);
```

```
tg = SimulinkRealTime.target
```

```
Target: TargetPC1
```

```
Connected          = Yes  
Application        = xpcosc  
Mode               = Real-Time Single-Tasking  
Status             = stopped  
CPUOverload       = none
```

```
ExecTime           = 0.0000  
SessionTime       = 171.7449  
StopTime           = 0.200000  
SampleTime        = 0.000250  
AvgTET            = NaN  
MinTET            = Inf  
MaxTET            = 0.000000  
ViewMode          = 0
```

```
TimeLog            = Vector(0)  
StateLog           = Matrix (0 x 2)  
OutputLog          = Matrix (0 x 2)  
TETLog            = Vector(0)  
MaxLogSamples      = 16666  
NumLogWraps        = 0  
LogMode            = Normal
```

```
Scopes             = No Scopes defined  
NumSignals         = 7  
ShowSignals        = off
```

```
NumParameters      = 7  
ShowParameters     = off
```

Add and configure host scope 1.

```
sc1 = addscope(tg, 'host', 1);  
addsignal(sc1, 4);  
addsignal(sc1, 5)
```

```
ans =
```

```
Simulink Real-Time Scope
```

```
Application        = xpcosc  
ScopeId           = 1  
Status            = Interrupted
```

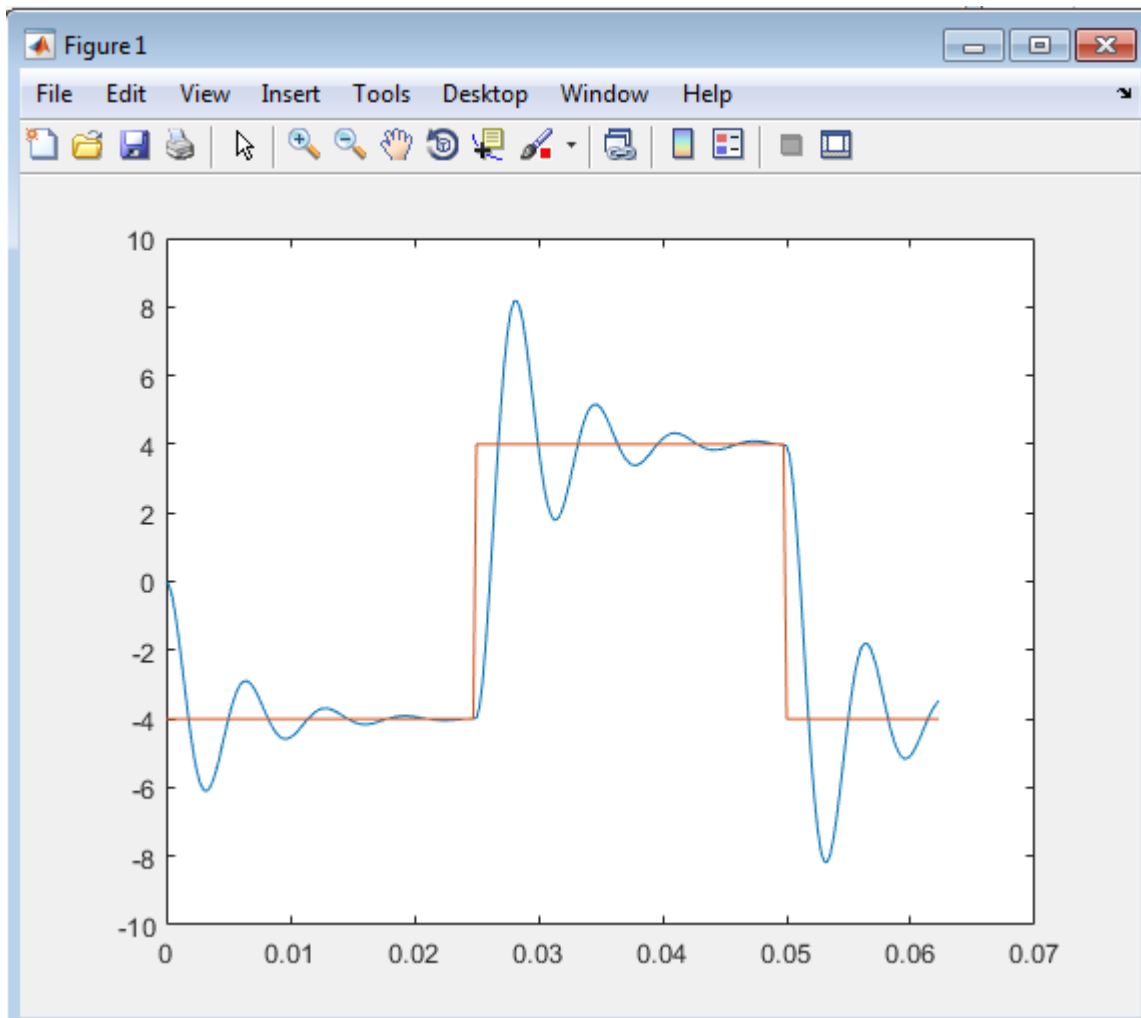
```
Type                = Host
NumSamples          = 250
NumPrePostSamples   = 0
Decimation          = 1
TriggerMode         = FreeRun
TriggerSignal       = 4 : Integrator1
TriggerLevel        = 0.000000
TriggerSlope        = Either
TriggerScope        = 1
TriggerSample       = 0
StartTime           = -1.000000
Data                = Matrix (250 x 2)
Time                = Matrix (250 x 1)
Signals             = 4 : Integrator1
                   5 : Signal Generator
```

Run the real-time application for 10 seconds.

```
tg.StopTime = 10;
start(sc1);
start(tg);
pause(10);
stop(tg);
stop(sc1);
```

Plot the result.

```
plot(sc1.Time,sc1.Data);
```



Unload the real-time application.

```
unload(tg)
```

```
Target: TargetPC1  
Connected = Yes
```

Application = loader

See Also

“Target Computer Commands” | Real-Time Application | Real-Time Application Properties | `SimulinkRealTime.target.getscope` | `SimulinkRealTime.target.remscope`

More About

- “Signal Tracing With a Host Scope in Freerun Mode”
- “Signal Tracing Using Software Triggering”
- “Signal Tracing Using Signal Triggering”
- “Signal Tracing Using Scope Triggering”
- “Pre- and Post-Triggering of a Host Scope”
- “Simulink Real-Time Scope Usage”
- “Host Scope Usage”

Introduced in R2014a

SimulinkRealTime.hostScope.addsignal

Add signals to host scope represented by scope object

Syntax

```
addsignal(scope_object_vector, signal_index_vector)
```

Arguments

<code>scope_object_vector</code>	Name of a single scope object or the name of a vector of scope objects.
<code>signal_index_vector</code>	For one signal, use a single number. For two or more signals, enclose numbers in brackets and separate with commas.

Description

`addsignal(scope_object_vector, signal_index_vector)` adds signals to a scope object. Specify the signals by their indices, which you can retrieve using the target object method `SimulinkRealTime.target.getsignalid`. If `scope_object_vector` has two or more scope objects, the same signals are assigned to each scope.

Before you can add a signal to a scope, you must stop the scope.

At the target computer command line, you can add multiple signals to the scope:

```
addsignal scope_index = signal_index, signal_index, . . .
```

Examples

The following examples use model `xpcosc`.

Add signals `Integrator1` and `Signal Generator` to scope object `sc1`.


```
tg = slrt;  
sc1 = addscope(tg, 'host', 1);  
s0 = getsignalid(tg, 'Signal Generator');  
s1 = getsignalid(tg, 'Integrator1');  
addsignal(sc1, [s0, s1]);
```

The scope object property **Signals** is updated to include the added signals. Type **sc1** to display the properties and values for scope **sc1**.

More About

- “Find Signal and Parameter Indexes”
- “Host Scope Usage”

See Also

“Target Computer Commands” | Real-Time Host Scope | Real-Time Application | Real-Time Application Properties | `SimulinkRealTime.hostScope.remsignal`

Introduced in R2014a

SimulinkRealTime.hostScope.remsignal

Remove signals from host scope represented by scope object

Syntax

```
remsignal(scope_object)  
remsignal(scope_object, signal_index_vector)
```

Arguments

<code>scope_object_vector</code>	Scope object or vector of scope objects. The target object methods <code>addscope</code> or <code>getscope</code> create scope objects.
<code>signal_index_vector</code>	Index numbers from the scope object property <code>Signals</code> . This argument is optional. If it is left out, all signals are removed.

Description

`remsignal(scope_object)` removes all signals from a scope object.

`remsignal(scope_object, signal_index_vector)` removes signals from a scope object. Specify the signals by their indices, which you can retrieve using the target object method `getsignalid`. If `scope_object` is a vector containing two or more scope objects, the same signals are removed from each scope.

Before you can remove a signal from a scope, you must stop the scope.

At the target computer command line, you can remove multiple signals from the scope:

```
remsignal scope_index = signal_index, signal_index, . . .
```

`signal_index` is optional. If it is left out, all signals are removed.

Examples

The following examples use model `xpcosc`.

Remove all signals from the scope represented by the scope object `sc1`:

```
tg = slrt;  
sc1 = getscope(tg,1);  
remsignal(sc1)
```

Remove signals `Integrator1` and `Signal Generator` from the scope on the target computer:

```
tg = slrt;  
sc1 = getscope(tg,1);  
s0 = getsignalid(tg,'Signal Generator');  
s1 = getsignalid(tg,'Integrator1');  
remsignal(sc1,[s0,s1])
```

More About

- “Find Signal and Parameter Indexes”
- “Host Scope Usage”

See Also

“Target Computer Commands” | Real-Time Host Scope | Real-Time Application | Real-Time Application Properties | `SimulinkRealTime.hostScope.addsignal`

Introduced in R2014a

SimulinkRealTime.hostScope.start

Start execution of host scope on target computer

Syntax

```
start(scope_object)
start(scope_object_vector)
start([scope_object1, scope_object2, . . . , scope_objectN])
```

Arguments

<code>scope_object</code>	Name of single vector object.
<code>scope_object_vector</code>	Name of vector of scope objects.

Description

`start(scope_object)` starts a scope on the target computer represented by a scope object on the development computer. Whether data acquisition starts depends on the trigger settings.

Before using this method, you must create a scope. To create a scope, use the target object method `addscope` or add Simulink Real-Time scope blocks to your Simulink model.

Alternative syntaxes are `start(scope_object_vector)` and `start([scope_object1, scope_object2, . . . , scope_objectN])`.

At the target computer command line, you can use the corresponding command:

```
startscope scope_index
startscope all
```

Examples

Start one scope with the scope object `sc1`:

```
tg = slrt;  
sc1 = getscope(tg,1)  
start(sc1)
```

Start two scopes, 1 and 2:

```
tg = slrt;  
somescope = getscope(tg,[1,2])  
start(somescope)
```

or

```
tg = slrt;  
sc1 = getscope(tg,1)  
sc2 = getscope(tg,2)  
start([sc1,sc2])
```

Start all scopes:

```
tg = slrt;  
allscopes = getscope(tg)  
start(allscopes)
```

More About

- “Trace Signals at Target Computer Command Line”
- “Host Scope Usage”

See Also

“Target Computer Commands” | Real-Time Host Scope | Real-Time Application | Real-Time Application Properties | `SimulinkRealTime.hostScope.stop`

Introduced in R2014a

SimulinkRealTime.hostScope.stop

Stop execution of host scope on target computer

Syntax

```
stop(scope_object)
stop(scope_object_vector)
stop([scope_object1, scope_object2, . . ., scope_objectN])
```

Arguments

<code>scope_object</code>	Name of single vector object.
<code>scope_object_vector</code>	Name of vector of scope objects.

Description

`stop(scope_object)` stops a scope on the target computer represented by a scope object on the development computer.

Alternative syntaxes are `stop(scope_object_vector)` and `stop([scope_object1, scope_object2, . . ., scope_objectN])`.

At the target computer command line, you can use the corresponding command:

```
stopscope scope_index
stopscope all
```

Examples

Stop one scope with the scope object `sc1`:

```
tg = slrt;
sc1 = getscope(tg,1)
stop(sc1)
```

Stop two scopes, 1 and 2:

```
tg = slrt;  
somescope = getscope(tg,[1,2])  
stop(somescope)
```

or

```
tg = slrt;  
sc1 = getscope(tg,1)  
sc2 = getscope(tg,2)  
stop([sc1,sc2])
```

Stop all scopes:

```
tg = slrt;  
allscopes = getscope(tg)  
stop(allscopes)
```

More About

- “Trace Signals at Target Computer Command Line”
- “Host Scope Usage”

See Also

“Target Computer Commands” | Real-Time Host Scope | Real-Time Application | Real-Time Application Properties | `SimulinkRealTime.hostScope.start`

Introduced in R2014a

SimulinkRealTime.hostScope.trigger

Software-trigger start of data acquisition for host scope

Syntax

```
trigger(scope_object_vector)
```

Arguments

<code>scope_object_vector</code>	Name of a single scope object, name of a vector of scope objects, or a list of scope object names in a vector form [<code>scope_object1</code> , <code>scope_object2</code>].
----------------------------------	---

Description

`trigger(scope_object_vector)` triggers the scope represented by the scope object to acquire the number of data points in the scope object property `NumSamples`.

If the scope object property `TriggerMode` has a value of `'Software'`, this function is the only way to trigger the scope. However, this function can be used on any scope, regardless of trigger mode setting. For example, if a scope did not trigger because the triggering criteria were not met, this function can be used to force the scope to trigger.

Examples

Using model `xpcosc`, set a single host scope to software trigger, trigger the acquisition of one set of samples, and plot the data on the host from the scope object properties `scope_object.Time` and `scope_object.Data`.

```
tg = slrt;  
tg.StopTime = Inf;  
sc1 = addscope(tg, 'host', 1);  
addsignal(sc1, 4)  
sc1.TriggerMode = 'software';
```



```
start(tg)
start(sc1)
trigger(sc1)
pause(0.5)
plot(sc1.Time, sc1.Data)
stop(sc1)
stop(tg)
```

More About

- “Trace Signals at Target Computer Command Line”
- “Host Scope Usage”

See Also

“Target Computer Commands” | Real-Time Host Scope | Real-Time Application | Real-Time Application Properties

Introduced in R2014a

Real-Time Target Scope

Display time-domain data on target computer

Description

Controls and accesses properties of target scopes.

The kernel acquires a data package and the scope displays the data on the target computer. Depending on the setting of `DisplayMode`, the data is displayed numerically or graphically by a redrawing, sliding, and rolling display.

The following lexical rules exist:

- Function names are case sensitive. Type the entire name.
- Property names are not case sensitive. You do not need to type the entire name, as long as the characters that you type are unique for the property.

You can invoke some of the scope object properties and functions from the target computer command line when you have loaded the real-time application.

Create Object

`SimulinkRealTime.target.addscope`

Properties

Use scope object properties to select signals that you want to acquire, set triggering modes, and access signal information from the real-time application.

To get the value of a readable scope object property from a scope object:

```
scope_object = getscope(target_object, scope_number);  
value = scope_object.scope_object_property
```

To get the Decimation of scope 3:

```
scope_object = getscope(tg, 3);
value = scope_object.Decimation
```

To set the value of a writable scope property from a scope object:

```
scope_object = getscope(target_object, scope_number);
scope_object.scope_object_property = new_value
```

To set the Decimation of scope 3:

```
scope_object = getscope(tg, 3);
scope_object.Decimation = 10
```

Not all properties are user-writable. For example, after you create the scope, property Type is not writable.

Target Scope Properties

DisplayMode — How target scope displays signals

'redraw' (default) | 'numerical' | 'rolling'

Indicates how a target scope displays the signals:

- 'redraw' — The scope plots signal values when the scope has acquired `numsamples` samples.
- 'numerical' — The scope displays signal values as text.
- 'rolling' — The scope plots signal values at every sample time.

The value 'sliding' will be removed in a future release. It behaves like value rolling.

Grid — Displays a grid on target screen

'on' (default) | 'off'

When 'on', displays a grid on the target screen.

YLimit — Range of y-axis values

'auto' (default) | numeric

Minimum and maximum y-axis limits. If **YLimit** is 'auto', the scope calculates the y-axis limits from the range of data values it is displaying.

Common Scope Properties

Application — Name of the real-time application associated with this scope object
character vector

Read-only name of the real-time application associated with this scope object.

Decimation — Samples to acquire

1 (default) | unsigned integer

Scope acquires every Decimationth sample.

NumPrePostSamples — Samples collected before or after a trigger event

0 (default) | integer

Number of samples collected before or after a trigger event. 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.

NumSamples — Number of contiguous samples captured

unsigned integer

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

The scope writes data samples into a memory buffer of size `NumSamples`. If the scope stops before capturing this number of samples, the scope writes zeroes after the collected data to the end of the buffer. Know what type of data you are collecting, because it is possible that your data contains zeroes.

ScopeId — Unique numeric index

unsigned integer

Read-only numeric index, unique for each scope.

Signals — Signal indexes to display on scope

unsigned integer vector

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

Status — State of scope acquisition

'Acquiring' | 'Ready for being Triggered' | 'Interrupted' | 'Finished'

Read-only state value:

- 'Acquiring' — The scope is acquiring data.
- 'Ready for being Triggered' — The scope is waiting for a trigger.
- 'Interrupted' — The scope is not running (interrupted).
- 'Finished' — The scope has finished acquiring data.

TriggerLevel — Signal trigger crossing value

numeric

If `TriggerMode` is 'Signal', this parameter indicates the value that 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.

TriggerMode — Scope trigger mode

'FreeRun' (default) | 'software' | 'signal' | 'scope'

Trigger mode for a scope:

- 'freerun' — The scope triggers on every sample time.
- 'software' — The scope triggers from the Command Window.
- 'signal' — The scope triggers when a designated signal changes state.
- 'scope' — The scope triggers when a designated scope triggers.

TriggerSample — Trigger sample for scope trigger

0 (default) | -1 | integer

If `TriggerMode` is 'Scope', then `TriggerSample` specifies on which sample of the triggering scope the current scope triggers.

For example, if `TriggerSample` is 0 (default), the current scope triggers on sample 0 (first sample acquired) of the triggering scope. In this case, the two scopes are synchronized with each other.

If `TriggerSample` is 1, the current scope triggers on sample 1 (second sample acquired) of the triggering scope. In this case, the two scopes have a one-sample offset.

Setting `TriggerSample` to -1 means that the current scope triggers at the end of the acquisition cycle of the triggering scope. In this case, the triggered scope acquires its first sample one sample after the last sample of the triggering scope.

TriggerScope — Scope for scope trigger

unsigned integer

If `TriggerMode` is `'Scope'`, this parameter identifies the scope to use for a trigger. To trigger a scope when another scope is triggered, set the slave scope property `TriggerScope` to the scope index of the master scope.

TriggerSignal — Signal for signal trigger

unsigned integer

If `TriggerMode` is `'Signal'`, this parameter identifies the block output signal to use for triggering the scope. Identify the signal with a signal index from the target object property `Signal`.

TriggerSlope — Trigger slope for signal trigger

'Either' (default) | 'Rising' | 'Falling'

If `TriggerMode` is `'Signal'`, `TriggerSlope` indicates the signal behavior that triggers the scope.

- `'Either'` — The signal triggers the scope when it crosses `triggerlevel` in either the rising or falling directions.
- `'Rising'` — The signal triggers the scope when it crosses `triggerlevel` in the rising direction.
- `'Falling'` — The signal triggers the scope when it crosses `triggerlevel` in the falling direction.

Type — Type of scope

'Host' (default) | 'Target' | 'File'

Read-only property that determines how the scope collects and displays its data:

- `'Host'` — The scope collects data on the target computer and displays it on the development computer.
- `'Target'` — The scope collects data on the target computer and displays it on the target computer monitor.
- `'File'` — The scope collects and stores data on the target computer.

Object Functions

`SimulinkRealTime.targetScope.addsignal` Add signals to target scope represented by scope object

<code>SimulinkRealTime.targetScope.remsignal</code>	Remove signals from target scope represented by scope object
<code>SimulinkRealTime.targetScope.start</code>	Start execution of target scope on target computer
<code>SimulinkRealTime.targetScope.stop</code>	Stop execution of target scope on target computer
<code>SimulinkRealTime.targetScope.trigger</code>	Software-trigger start of data acquisition for target scope

Examples

Build and Run Real-Time Application with Target Scope

Build and download `xpcosc` and execute the real-time application with a target scope.

Open, build, and download the real-time application.

```
ex_model = 'xpcosc';
open_system(ex_model);
rtwbuild(ex_model);
tg = SimulinkRealTime.target
```

```
Target: TargetPC1
  Connected          = Yes
  Application        = xpcosc
  Mode               = Real-Time Single-Tasking
  Status             = stopped
  CPUOverload        = none

  ExecTime           = 0.0000
  SessionTime        = 158.0022
  StopTime           = 0.200000
  SampleTime         = 0.000250
  AvgTET             = NaN
  MinTET             = Inf
  MaxTET             = 0.000000
  ViewMode           = 0

  TimeLog            = Vector(0)
  StateLog           = Matrix (0 x 2)
  OutputLog          = Matrix (0 x 2)
  TETLog             = Vector(0)
```

```
MaxLogSamples      = 16666
NumLogWraps        = 0
LogMode            = Normal

Scopes             = No Scopes defined
NumSignals         = 7
ShowSignals        = off

NumParameters      = 7
ShowParameters     = off
```

Add and configure target scope 1.

```
sc1 = addscope(tg, 'target', 1);
addsignal(sc1, 4);
addsignal(sc1, 5)
```

```
ans =
```

```
Simulink Real-Time Scope
Application        = xpcosc
ScopeId           = 1
Status            = Interrupted
Type              = Target
NumSamples        = 250
NumPrePostSamples = 0
Decimation        = 1
TriggerMode       = FreeRun
TriggerSignal     = 4 : Integrator1
TriggerLevel      = 0.000000
TriggerSlope      = Either
TriggerScope      = 1
TriggerSample     = 0
DisplayMode       = Redraw (Graphical)
YLimit            = Auto
Grid              = on
Signals           = 4 : Integrator1
                  5 : Signal Generator
```

Run the real-time application for 10 seconds.

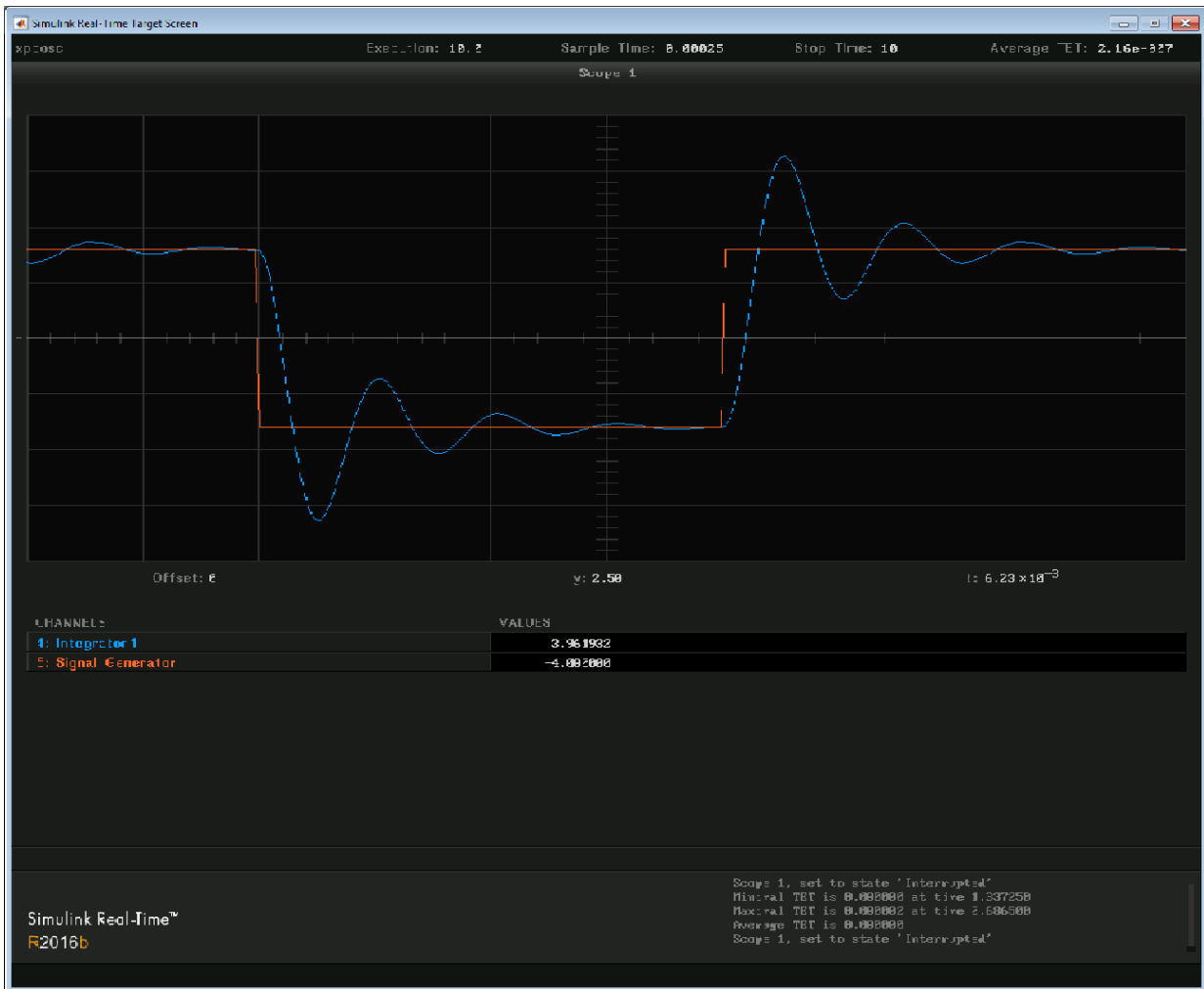
```
tg.StopTime = 10;
start(sc1);
start(tg);
pause(10);
```



```
stop(tg);
stop(sc1);
```

View the target screen on the development computer.

```
viewTargetScreen(tg);
```



Unload the real-time application.

```
unload(tg)
```

```
Target: TargetPC1  
  Connected           = Yes  
  Application         = loader
```

See Also

“Target Computer Commands” | Real-Time Host Scope | Real-Time Application | Real-Time Application Properties | Real-Time File Scope | SimulinkRealTime.target.getscope | SimulinkRealTime.target.remscope

More About

- “Signal Tracing With a Target Scope”
- “Simulink Real-Time Scope Usage”
- “Target Scope Usage”

Introduced in R2014a

SimulinkRealTime.targetScope.addsignal

Add signals to target scope represented by scope object

Syntax

```
addsignal(scope_object_vector, signal_index_vector)
```

Arguments

<code>scope_object_vector</code>	Name of a single scope object or the name of a vector of scope objects.
<code>signal_index_vector</code>	For one signal, use a single number. For two or more signals, enclose numbers in brackets and separate with commas.

Description

`addsignal(scope_object_vector, signal_index_vector)` adds signals to a scope object. Specify the signals by their indices, which you can retrieve using the target object method `SimulinkRealTime.target.getsignalid`. If `scope_object_vector` has two or more scope objects, the same signals are assigned to each scope.

Before you can add a signal to a scope, you must stop the scope.

At the target computer command line, you can add multiple signals to the scope:

```
addsignal scope_index = signal_index, signal_index, . . .
```

Examples

The following examples use model `xpcosc`.

Add signals `Integrator1` and `Signal Generator` to scope object `sc1`.

```
tg = slrt;  
sc1 = addscope(tg, 'target', 1);  
s0 = getsignalid(tg, 'Signal Generator');  
s1 = getsignalid(tg, 'Integrator1');  
addsignal(sc1, [s0, s1]);
```

The scope object property **Signals** is updated to include the added signals. Type **sc1** to display the properties and values for scope **sc1**.

More About

- “Find Signal and Parameter Indexes”
- “Target Scope Usage”

See Also

“Target Computer Commands” | Real-Time Target Scope | Real-Time Application | Real-Time Application Properties | `SimulinkRealTime.targetScope.remsignal`

Introduced in R2014a

SimulinkRealTime.targetScope.remsignal

Remove signals from target scope represented by scope object

Syntax

```
remsignal(scope_object)  
remsignal(scope_object, signal_index_vector)
```

Arguments

<code>scope_object_vector</code>	Scope object or vector of scope objects. The target object methods <code>addscope</code> or <code>getscope</code> create scope objects.
<code>signal_index_vector</code>	Index numbers from the scope object property <code>Signals</code> . This argument is optional. If it is left out, all signals are removed.

Description

`remsignal(scope_object)` removes all signals from a scope object.

`remsignal(scope_object, signal_index_vector)` removes signals from a scope object. Specify the signals by their indices, which you can retrieve using the target object method `getsignalid`. If `scope_object` is a vector containing two or more scope objects, the same signals are removed from each scope.

Before you can remove a signal from a scope, you must stop the scope.

At the target computer command line, you can remove multiple signals from the scope:

```
remsignal scope_index = signal_index, signal_index, . . .
```

`signal_index` is optional. If it is left out, all signals are removed.

Examples

The following examples use model `xpcosc`.

Remove all signals from the scope represented by the scope object `sc1`.

```
tg = slrt;  
sc1 = getscope(tg,1);  
remsignal(sc1)
```

Remove signals `Integrator1` and `Signal Generator` from the scope on the target computer.

```
tg = slrt;  
sc1 = getscope(tg,1);  
s0 = getsignalid(tg,'Signal Generator');  
s1 = getsignalid(tg,'Integrator1');  
remsignal(sc1,[s0,s1])
```

More About

- “Find Signal and Parameter Indexes”
- “Target Scope Usage”

See Also

“Target Computer Commands” | `Real-Time Target Scope` | `Real-Time Application` | `Real-Time Application Properties` | `SimulinkRealTime.targetScope.addsignal`

Introduced in R2014a

SimulinkRealTime.targetScope.start

Start execution of target scope on target computer

Syntax

```
start(scope_object)
start(scope_object_vector)
start([scope_object1, scope_object2, . . ., scope_objectN])
```

Arguments

<code>scope_object</code>	Name of single vector object.
<code>scope_object_vector</code>	Name of vector of scope objects.

Description

`start(scope_object)` starts a scope on the target computer represented by a scope object on the development computer. Whether data acquisition starts depends on the trigger settings.

Before using this method, you must create a scope. To create a scope, use the target object method `addscope` or add Simulink Real-Time scope blocks to your Simulink model.

Alternative syntaxes are `start(scope_object_vector)` and `start([scope_object1, scope_object2, . . ., scope_objectN])`.

At the target computer command line, you can use the corresponding command:

```
startscope scope_index
startscope all
```

Examples

Start one scope with the scope object `sc1`.

```
tg = slrt;  
sc1 = getscope(tg,1)  
start(sc1)
```

Start two scopes, 1 and 2.

```
tg = slrt;  
somescope = getscope(tg,[1,2])  
start(somescope)
```

or

```
tg = slrt;  
sc1 = getscope(tg,1)  
sc2 = getscope(tg,2)  
start([sc1,sc2])
```

Start all scopes:

```
tg = slrt;  
allscopes = getscope(tg)  
start(allscopes)
```

More About

- “Trace Signals at Target Computer Command Line”
- “Target Scope Usage”

See Also

“Target Computer Commands” | Real-Time Target Scope | Real-Time Application | Real-Time Application Properties | `SimulinkRealTime.targetScope.stop`

Introduced in R2014a

SimulinkRealTime.targetScope.stop

Stop execution of target scope on target computer

Syntax

```
stop(scope_object)
stop(scope_object_vector)
stop([scope_object1, scope_object2, . . ., scope_objectN])
```

Arguments

<code>scope_object</code>	Name of single vector object.
<code>scope_object_vector</code>	Name of vector of scope objects.

Description

`stop(scope_object)` stops a scope on the target computer represented by a scope object on the development computer.

Alternative syntaxes are `stop(scope_object_vector)` and `stop([scope_object1, scope_object2, . . ., scope_objectN])`.

At the target computer command line, you can use the corresponding command:

```
stopscope scope_index
stopscope all
```

Examples

Stop one scope with the scope object `sc1`.

```
tg = slrt;
sc1 = getscope(tg,1)
stop(sc1)
```

Stop two scopes, 1 and 2.

```
tg = slrt;  
somescope = getscope(tg,[1,2])  
stop(somescope)
```

or

```
tg = slrt;  
sc1 = getscope(tg,1)  
sc2 = getscope(tg,2)  
stop([sc1,sc2])
```

Stop all scopes:

```
tg = slrt;  
allscopes = getscope(tg)  
stop(allscopes)
```

More About

- “Trace Signals at Target Computer Command Line”
- “Target Scope Usage”

See Also

“Target Computer Commands” | Real-Time Target Scope | Real-Time Application | Real-Time Application Properties | `SimulinkRealTime.targetScope.start`

Introduced in R2014a

SimulinkRealTime.targetScope.trigger

Software-trigger start of data acquisition for target scope

Syntax

```
trigger(scope_object_vector)
```

Arguments

<code>scope_object_vector</code>	Name of a single scope object, name of a vector of scope objects, or a list of scope object names in a vector form [<code>scope_object1</code> , <code>scope_object2</code>].
----------------------------------	---

Description

`trigger(scope_object_vector)` triggers the scope represented by the scope object to acquire the number of data points in the scope object property `NumSamples`.

If the scope object property `TriggerMode` has a value of `'Software'`, this function is the only way to trigger the scope. However, this function can be used on any scope, regardless of trigger mode setting. For example, if a scope did not trigger because the triggering criteria were not met, this function can be used to force the scope to trigger.

Examples

Using model `xpcosc`, set a single target scope to software trigger, trigger the acquisition of one set of samples, and display the data on the target display.

```
tg = slrt;  
tg.StopTime = Inf;  
sc1 = addscope(tg, 'target', 1);  
addsignal(sc1, 4)  
sc1.TriggerMode = 'software';  
start(tg)
```

```
start(sc1)
trigger(sc1)
pause(0.5)
stop(sc1)
stop(tg)
```

More About

- [“Find Signal and Parameter Indexes”](#)
- [“Target Scope Usage”](#)

See Also

[“Target Computer Commands”](#) | [Real-Time Target Scope](#) | [Real-Time Application](#) | [Real-Time Application Properties](#)

Introduced in R2014a