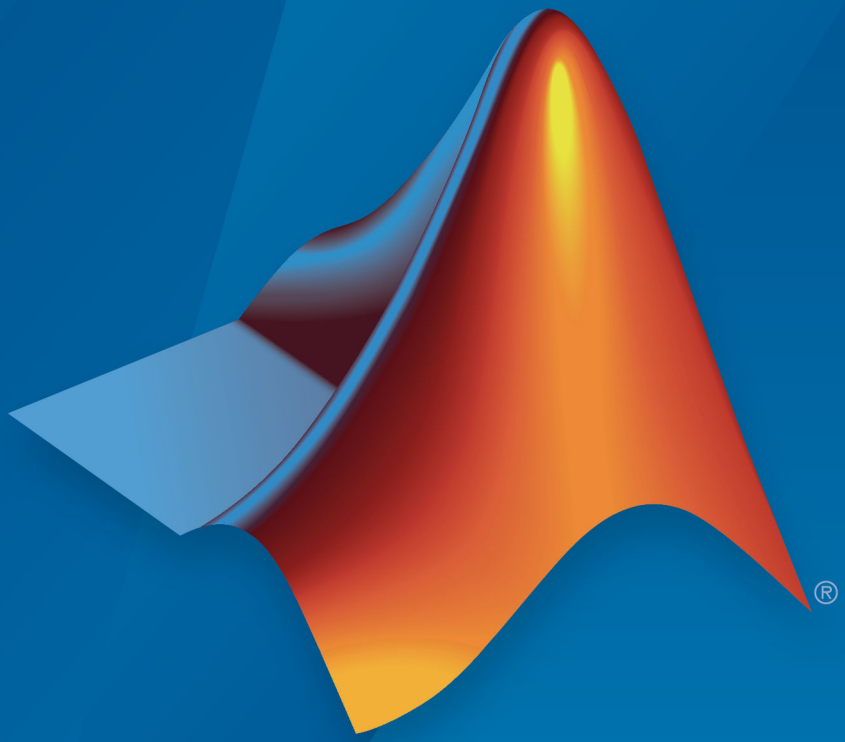
Simulink® Real-Time™

API Guide



MATLAB® & SIMULINK®

R2019b

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

Sales and services: `www.mathworks.com/sales_and_services`

User community: `www.mathworks.com/matlabcentral`

Technical support: `www.mathworks.com/support/contact_us`

Phone: 508-647-7000

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

Simulink® Real-Time™ API Guide

© COPYRIGHT 2002–2019 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)
March 2017	Online only	Revised for Version 6.6 (Release 2017a)
September 2017	Online only	Revised for Version 6.7 (Release 2017b)
March 2018	Online only	Revised for Version 6.8 (Release 2018a)
September 2018	Online only	Revised for Version 6.9 (Release 2018b)
March 2019	Online only	Revised for Version 6.10 (Release 2019a)
September 2019	Online only	Revised for Version 6.11 (Release 2019b)

Contents

1 Introduction

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**

2 Simulink Real-Time API for Microsoft .NET Framework

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**

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

4 Simulink Real-Time API for C

Using the C API **4-2**

5 Simulink Real-Time API Reference for C

6 MATLAB API

1

Introduction

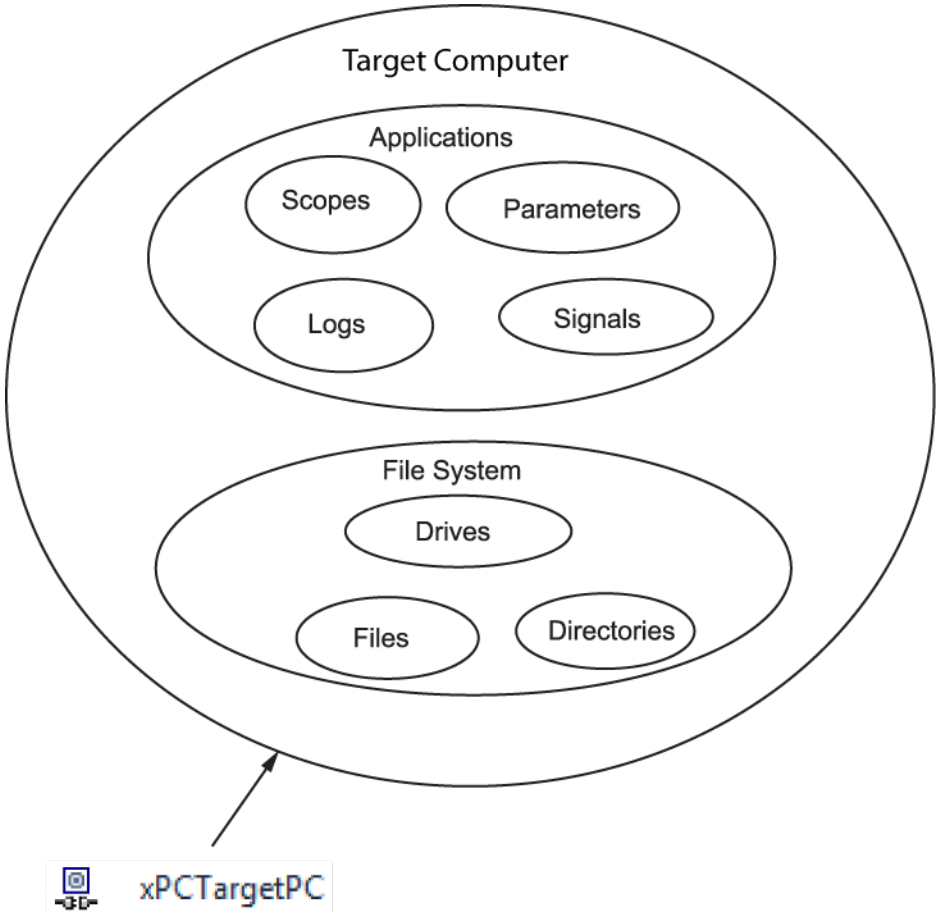
- “ Simulink Real-Time API for Microsoft .NET Framework” on page 1-2
- “ Simulink Real-Time C API” on page 1-7
- “C API Error Messages” on page 1-8

Simulink Real-Time API for Microsoft .NET Framework

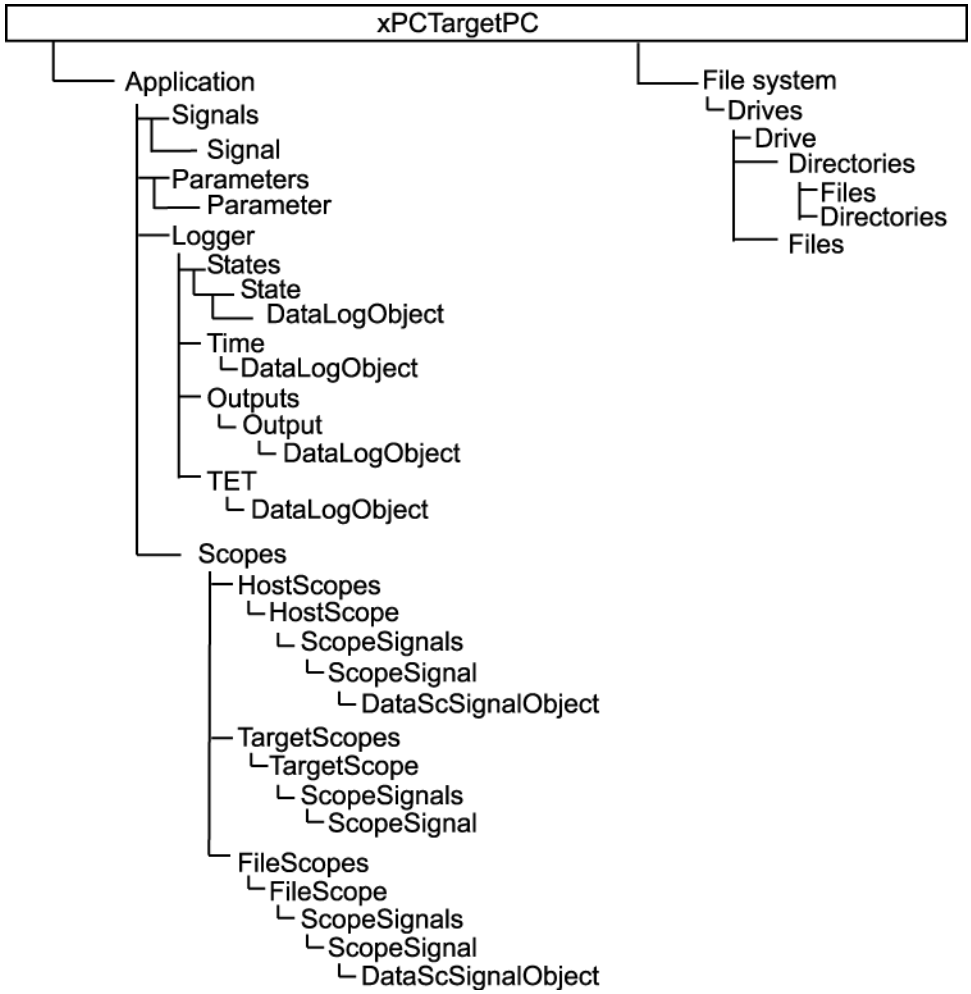
The Simulink Real-Time API for Microsoft .NET Framework consists of objects arranged in hierarchical order. Each of these objects has functions and properties that allow you to manipulate and interact with the API. The API provides various object types, including objects for the target computer, real-time applications, scopes, and the file system. You can use these API functions from languages and custom programs that support managed code, such as Microsoft Visual Studio®, Windows® PowerShell, and 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 session information. `xPCTargetPC` provides 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. You 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. Use the Simulink Real-Time C API to build custom programs for development environments such as Microsoft Foundation Class Library/ Active Template Library (MFC/ATL) and third-party product APIs such as 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. Build this file along with the custom Simulink Real-Time C API program.

The Simulink Real-Time C API consists of blocking functions. 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, you can call the API functions from non-C languages, such as C++ and Java.

Note Refer to the compiler documentation of the non-C language for a description of how to access C functions from a library DLL. To access the Simulink Real-Time C API DLL, follow these directions. For information about working with 64-bit applications in Visual Studio, see 64-bit Applications (Microsoft docs).

C API Error Messages

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

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

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

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

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

Simulink Real-Time API for Microsoft .NET Framework

- “Using the Simulink Real-Time API for Microsoft .NET Framework” on page 2-2
- “ Simulink Real-Time .NET API Application Creation” on page 2-4
- “ Simulink Real-Time .NET API Application Distribution” on page 2-6
- “ Simulink Real-Time .NET API Client Application Examples” on page 2-7

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`.

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 calls the Simulink Real-Time API for the .NET Framework, use a third-party development environment 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 `slrtpingtarget` from the Command Window to check whether the development and target computers are 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 that the client 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 create .NET 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.
- `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.

3

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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 ILIST of xPCHostScopeSignal objects.

Exception

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

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

xPCTargetPC.Connect

Establish connection with 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.

Introduced in R2011b

xPCTargetPC.ConnectAsync

Asynchronous request for target computer connection

Syntax

```
public void ConnectAsync()
```

Description

Class: `xPCTargetPC Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Exception

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

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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`	*`DevelDestFileName`* is empty, contains only white spaces, or contains invalid characters.
`ArgumentNullException`	`DevelDestFileName` is NULL reference.
`NotSupportedException`	*`DevelDestFileName`* contains a colon (`:`) in the middle of the character string.

Exception	Condition
`PathTooLongException`	The specified path, file name, or both in *`DevelDestFileName`* exceed the system-defined maximum length. For example, on Windows platforms, path names must be fewer than 248 characters. File names must be fewer than 260 characters.
`SecurityException`	Caller does not have required permission.
`UnauthorizedAccess-Exception`	System does not allow access to *`DevelDestFileName`*.
`xPCException`	When problem occurs, query `xPCException` object `Reason` property.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2016a

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

xPCTargetPC.DisconnectAsync

Asynchronous request to disconnect from target computer

Syntax

```
public void DisconnectAsync()
```

Description

Class: `xPCTargetPC Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: `C#`

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

Exception

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

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

xPCDataFileScSignalObject.GetDataAsync

Asynchronously copy file scope signal data from target computer

Syntax

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

Description

Class: `xPCDataFileScSignalObject Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

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

Exception

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

Introduced in R2011b

xPCDataHostScSignalObject.GetDataAsync

Asynchronously copy host scope signal data from target computer

Syntax

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

Description

Class: `xPCDataHostScSignalObject Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

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

Exception

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

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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 xPCFileSystemInfo entries. These entries represent the files and subfolders in a folder.

Introduced in R2011b

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.

Introduced in R2011b

xPCParameter.GetParamAsync

Asynchronous request to get parameter values from target computer

Syntax

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

Description

Class: `xPCParameter Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

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

Exception

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

Introduced in R2011b

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.

Introduced in R2011b

xPCSignals.GetSignals

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

Syntax

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

Description

Class: `xPCSignals Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public IList<xPCSignal> GetSignals(string[] arrayofBlockPath)` returns list of xPCSignal objects specified by array of signal identifiers. This method creates an ILIST of xPCSignal objects with an array of *`blockpath`*s. *`arrayofBlockPath`* is an array of 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.

Introduced in R2011b

xPCSignals.GetSignalsValue

Vector of signal values from array

Syntax

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

Description

Class: `xPCSignals Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

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

Exception

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

Introduced in R2011b

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.

Introduced in R2011b

xPCTargetPC.Load

Load real-time application onto target computer

Syntax

```
public xPCApplication Load()
public xPCApplication Load(string AppFileName)
```

Description

Class: `xPCTargetPC Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

`public xPCApplication Load(string AppFileName)` loads *`AppFileName`* onto the target computer. *`AppFileName`* is a character string that specifies the full path name, without file extension, to the real-time application that you are loading on the target computer. This method returns an xPCApplication object.

Exception

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

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

Introduced in R2011b

xPCTargetPC.LoadAsync

Asynchronous request to load real-time application onto target computer

Syntax

```
public void LoadAsync()
```

Description

Class: `xPCTargetPC Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Exception

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

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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 canceling a property value change.

Properties

Properties	C# Declaration Syntax	Description
`Cancel`	`public bool Cancel {get; set;}`	Get or set value indicating whether 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.

Introduced in R2012a

ConnectCompletedEventArgs Class

`xPCTargetPC.ConnectCompleted` event data

Syntax

```
public class ConnectCompletedEventArgs : AsyncCompletedEventArgs
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Properties

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

Introduced in R2012a

DisconnectCompletedEventArgs Class

`xPCTargetPC.DisconnectCompleted` event data

Syntax

```
public class DisconnectCompletedEventArgs : AsyncCompletedEventArgs
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Properties

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

Introduced in R2012a

GetDataCompletedEventArgs Class

`GetDataCompleted` event data

Syntax

```
public class GetDataCompletedEventArgs : AsyncCompletedEventArgs
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Properties

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

Introduced in R2012a

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 canceled.
`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.

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

GetHostScSignalDataObjectCompletedEventArgs Class

`xPCDataHostScSignalObject.DataObjectCompleted` event data

Syntax

```
public class GetHostScSignalDataObjectCompletedEventArgs :
GetDataCompletedEventArgs
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Properties

Properties	C# Declaration Syntax	Description
`Cancelled`	`public bool Cancelled {get;}`	Get value that indicates if an asynchronous operation has been canceled.
`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.

Properties	C# Declaration Syntax	Description
`IsScopeSignal`	`public bool IsScopeSignal {get;}`	Get if signal is a scope signal (`true`) or a time signal (`false`).
`ScopeSignalObject`	`public xPCScopeSignal ScopeSignalObject {get;}`	Get reference to parent xPCHostScopeSignal object
`State`	`public Object State {get;}`	Optional. Get user-supplied state object.
`UserState`	`public Object UserState {get;}`	Get unique identifier for asynchronous task.

GetLogDataCompletedEventArgs Class

`xPCDataLoggingObject.GetDataCompleted` event data

Syntax

```
public class GetLogDataCompletedEventArgs :
GetDataCompletedEventArgs
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Properties

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

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

Introduced in R2012a

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 canceled.
`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.

Introduced in R2012a

LoadCompletedEventArgs Class

`xPCTargetPC.LoadCompleted` event data

Syntax

```
public class LoadCompletedEventArgs : AsyncCompletedEventArgs
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Properties

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

Introduced in R2012a

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.

Introduced in R2012a

RebootCompletedEventArgs Class

`xPCTargetPC.RebootCompleted` event data

Syntax

```
public class RebootCompletedEventArgs : AsyncCompletedEventArgs
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Properties

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

Introduced in R2012a

SetParamCompletedEventArgs Class

`xPCParameter.SetParamCompleted` event data

Syntax

```
public class SetParamCompletedEventArgs : AsyncCompletedEventArgs
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Properties

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

Introduced in R2012a

UnloadCompletedEventArgs Class

`xPCTargetPC.UnloadCompleted` event data

Syntax

```
public class UnloadCompletedEventArgs : AsyncCompletedEventArgs
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Properties

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

Introduced in R2012a

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

Events	Description
`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 CPU overload flag.	`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.

Properties	C# Declaration Syntax	Description	Exception
`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 long it took to achieve the TET time.	`xPCException` — When problem occurs, query xPCException object `Reason` property.
`MinimumTeT`	`public double MinimumTeT {get;}`	Get the minimum 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.	

Properties	C# Declaration Syntax	Description	Exception
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.
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.	

Introduced in R2011b

xPCAppLogger Class

Access to real-time application loggers

Syntax

```
public class xPCAppLogger : xPCApplicationObject
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Properties

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

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

Introduced in R2011b

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
`xPCDataFileSc-SignalObject.GetData`	Copy file scope signal data from target computer
`xPCDataFileSc-SignalObject.-GetDataAsync`	Asynchronously copy file scope signal data from target computer

Events

Event	Description
`xPCDataFileSc-SignalObject.-GetDataCompleted`	Event when `xPCDataFileScSignalObject.GetDataAsync` is complete

Properties

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

Introduced in R2011b

xPCDataHostScSignalObject Class

Object that holds logged host scope signal data

Syntax

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

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

accesses an object that holds logged host scope signal data.

Methods

Method	Description
`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	public int Decimation {get; set;}	If 1, acquire every sample in a scope window. Otherwise, acquire every *n*th sample in a scope window.
NumSamples	public int NumSamples {get; set;}	Get or set number of contiguous samples captured during the acquisition of a data package. 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 is possible that your data contains zeroes.
ScopeSignal-Object	public xPCHostScopeSignal ScopeSignalObject {get;}	Get parent scope signal xPCHostScopeSignal object.
Startindex	public int StartIndex {get; set;}	Get and set the index of the first sample that you retrieve from the log.

Introduced in R2011b

xPCDataLoggingObject Class

Object that holds logged data

Syntax

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

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Methods

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

Events

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

Properties

Property	C# Declaration Syntax	Description
Decimation	public int Decimation {get; set;}	A number *n*, where every *n*th sample is acquired in a scope window.
LogId	public int LogId {get;}	
NumSamples	public int NumSamples {get; set;}	Get or set number of contiguous samples captured during the acquisition of a data package.
Startindex	public int StartIndex {get; set;}	Get and set the index of the first sample that you retrieve from the log.

Introduced in R2011b

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

Methods

Method	Description
`xPCDirectoryInfo.-Create`	Create folder
`xPCDirectoryInfo.-Delete`	Delete empty xPCDirectoryInfo object

Method	Description
xPCDirectoryInfo.-GetDirectories	Subfolders of current folder
xPCDirectoryInfo.-GetFiles	File list from current folder
xPCDirectoryInfo.-GetFileSystemInfos	File system information for files and subfolders in folder

Properties

Property	C# Declaration Syntax	Description	Exception
CreationTime	public override DateTime CreationTime {get;}	Get creation time of the current FileSystemInfo object.	xPCException — When problem occurs, query xPCException object Reason property.
Exists	public override bool Exists {get;}	Get a Boolean value that indicates the existence of the folder. A value of 1 indicates that the folder exists, 0 indicates that it does not.	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.

Property	C# Declaration Syntax	Description	Exception
`Root`	`public xPCDirectoryInfo Root {get;}`	Get the root portion of a path.	`xPCException` — When problem occurs, query xPCException object `Reason` property.

Introduced in R2011b

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 FAT-32.	`xPCException` — When problem occurs, query xPCException object `Reason` property.
`DriveType`	`public slrtDriveType DriveType {get;}`	Get drive type, such as `DRIVE_REMOVABLE`, `DRIVE_FIXED`, or `DRIVE_RAMDISK`.	`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.

Introduced in R2011b

xPCException Class

Information for `xPCException`

Syntax

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

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Constructor

Constructor	Description
`xPCException`	Construct new instance of xPCException class

Properties

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

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

Introduced in R2011b

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 at most 128 files open on the target computer at the same time.
- The largest single file that you can create on the target computer 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 in the operating system on the target computer can have a maximum of 260 characters. If the file name is longer than eight-dot-three format (eight character file name, period, three character extension), the operating system represents the file name in truncated form (for example, six characters followed by `'~1'`). MATLAB commands can access the file using the fully qualified file name or the truncated representation of the name. Some block parameters, such as the Scope block `filename` parameter, require 8.3 format for the file name.
- Do not write data to the `private` folder on your target computer. It is reserved for Simulink Real-Time internal use.

Constructor

Constructor	Description
`xPCFileInfo`	Construct new instance of xPCFileInfo class

Methods

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

Properties

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

Introduced in R2011b

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 at most 128 files open on the target computer at the same time.
- The largest single file that you can create on the target computer 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 in the operating system on the target computer can have a maximum of 260 characters. If the file name is longer than eight-dot-three format (eight character file name, period, three character extension), the operating system represents the file name in truncated form (for example, six characters followed by `'~1'`). MATLAB commands can access the file using the fully qualified file name or the truncated representation of the name. Some block parameters, such as the Scope block `filename` parameter, require 8.3 format for the file name.
- Do not write data to the `private` folder on your target computer. It is reserved for Simulink Real-Time internal use.

Methods

The xPCFileScope class inherits methods from `xPCScope Class`.

Events

The xPCFileScope class inherits events from `xPCScope Class`.

Properties

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

Property	C# Declaration Syntax	Description	Exception
`AutoRestart`	`public bool AutoRestart {get; set;}`	Get or set the file scope autorestart setting. `AutoRestart` is a Boolean. Values are `'on'` and `'off'`.	`xPCException` — When problem occurs, query xPCException object `Reason` property.
`DataTime-Object`	`public xPCDataHostScSignalObject DataTimeObject {get;}`	Get data time object.	`xPCException` — When problem occurs, query xPCException object `Reason` property.
`DynamicMode`	`public bool DynamicMode {get; set;}`	Get or set ability to create multiple log files for file scopes. Values are `'on'` and `'off'`. By default, the value is `'off'`.	`xPCException` — When problem occurs, query xPCException object `Reason` property.
`FileMode`	`public SCFILEMODE FileMode {get; set;}`	Get or set write mode of file. See xPCFileMode Enumerated Data Type.	`xPCException` — When problem occurs, query xPCException object `Reason` property.
`FileName`	`public string FileName {get; set;}`	Get or set file name for scope.	

Property	C# Declaration Syntax	Description	Exception
MaxWrite-FileSize	public uint MaxWriteFileSize {get; set;}	Get or set the maximum file size in bytes allowed before incrementing to the next file. When the size of a log file reaches MaxWriteFileSize, the software creates the next numbered file name. It continues logging data, incrementing to the next file as required, until it reaches the highest log file number you specified. If the software cannot create additional log files, it overwrites the first log file. This value must be a multiple of WriteSize. Default is 536870912.	xPCException — When problem occurs, query xPCException object Reason property.
Signals	public xPCTarget-ScopeSignalCollection Signals {get;}	Get collection of file scope signals (xPCFileScope-SignalCollection) assigned to this scope object.	
Trigger-Signal	public xPCTgtScopeSignal TriggerSignal {get; set;}	Get or set file scope signal (xPCFileScopeSignal) used to trigger the scope.	xPCException — When problem occurs, query xPCException object Reason property.

Property	C# Declaration Syntax	Description	Exception
`WriteSize`	`public int WriteSize {get; set;}`	Get or set the unit number of bytes for memory buffer writes. The memory buffer accumulates data in multiples of write size. *`WriteSize`* must be multiple of 512.	`xPCException` — When problem occurs, query xPCException object `Reason` property.

Introduced in R2011b

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

Introduced in R2011b

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.

Introduced in R2011b

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 Item[string blkpath] {get;}`	Get xPCFileScopeSignal object from signal name (*`blkpath`*). *`blkpath`* is the signal name that represents a signal object added to its parent xPCHostScope object. This property returns the file scope signal object as type xPCFileScopeSignal.	`xPCException` — When problem occurs, query xPCException object `Reason` property.

Introduced in R2011b

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 at most 128 files open on the target computer at the same time.
- The largest single file that you can create on the target computer 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 in the operating system on the target computer can have a maximum of 260 characters. If the file name is longer than eight-dot-three format (eight character file name, period, three character extension), the operating system represents the file name in truncated form (for example, six characters followed by `'~1'`). MATLAB commands can access the file using the fully qualified file name or the truncated representation of the name. Some block parameters, such as the Scope block `filename` parameter, require 8.3 format for the file name.
- Do not write data to the `private` folder on your target computer. It is reserved for Simulink Real-Time internal use.

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.

Introduced in R2011b

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 at most 128 files open on the target computer at the same time.
- The largest single file that you can create on the target computer 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 in the operating system on the target computer can have a maximum of 260 characters. If the file name is longer than eight-dot-three format (eight character file name, period, three character extension), the operating system represents the file name in truncated form (for example, six characters followed by `'~1'`). MATLAB commands can access the file using the fully qualified file name or the truncated representation of the name. Some block parameters, such as the Scope block `filename` parameter, require 8.3 format for the file name.
- Do not write data to the `private` folder on your target computer. It is reserved for Simulink Real-Time internal use.

Methods

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

Introduced in R2011b

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	`public abstract bool Exists {get;}`	Get value that indicates existence of file or folder.
Extension	`public string Extension {get;}`	Get character string that represents file extension.
FullName	`public virtual string FullName {get;}`	Get full path name of file or folder.
Name	`public abstract string Name {get;}`	Get name of folder.

Introduced in R2011b

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
`DataTime-Object`	`public xPCDataHostSc-SignalObject DataTimeObject {get;}`	Get host scope time data object xPCDataHost-ScSignalObject associated with this scope.	
`Signals`	`public xPCTarget-ScopeSignal-Collection Signals {get;}`	Get collection of host scope signals (xPCHost-ScopeSignalCollection) assigned to this scope object.	
`Trigger-Signal`	`public xPCTgtScope-Signal TriggerSignal {get; set;}`	Get or set host scope signal (xPCHostScope-Signal) used to trigger the scope.	`xPCException` — When problem occurs, query xPCException object `Reason` property.

Introduced in R2011b

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

Introduced in R2011b

xPCHostScopeSignal Class

Access to host scope signals

Syntax

```
public class xPCHostScopeSignal : xPCScopeSignal
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Properties

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

Introduced in R2011b

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 Item[string blkpath] {get;}`	Get xPCHostScopeSignal object from signal name (*`blkpath`*). *`blkpath`* is the signal name that represents a signal object added to its parent xPCHostScope object. This property returns the file scope signal object as type xPCHostScopeSignal.	`xPCException` — When problem occurs, query xPCException object `Reason` property.

Introduced in R2011b

xPCLog Class

Base data logging class

Syntax

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

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Properties

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

Introduced in R2011b

xPCOutputLogger Class

Access to output logger

Syntax

```
public class xPCOutputLogger : xPCLog
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Properties

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

Properties	C# Declaration Syntax	Description
`DataLoggingObjects`	`public IList<xPCDataLoggingObject> DataLoggingObjects {get;}`	Get ILIST of application data logging objects.
`IsEnabled`	`public override bool IsEnabled {get;}`	Get whether to enable or disable logging. Overrides xPCLog.IsEnabled.

Properties	C# Declaration Syntax	Description
`Item`	`public xPCDataLoggingObject Item[int index] {get;}`	Get xPCDataLogging object specified by index (*`index`*). *`index`* is the index to the specified logging output. This property returns an object of type xPCDataLoggingObject.
`NumOutputs`	`public int NumOutputs {get;}`	Return a reference to the xPCOutputLogger object.

Introduced in R2011b

xPCParameter Class

Single run-time tunable parameter

Syntax

```
public class xPCParameter : xPCApplicationNotficationObject
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Methods

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

Events

Event	Description
xPCParameter.-GetParamCompleted	Event when xPCParameter.GetParamAsync is complete
xPCParameter.-SetParamCompleted	Event when xPCParameter.SetParamAsync is complete

Properties

Property	C# Declaration Syntax	Description	Exception
BlockPath	public string BlockPath {get;}	Get the full block path name of the parameter for an instance of an xPCParameter object.	
DataType	public string DataType {get;}	Get the Simulink type, as a 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 numeric index (identifier) that maps to an instance of an xPCParameter object.	
Rank	public int Rank {get;}	Get the number of dimensions of the parameter	

Property	C# Declaration Syntax	Description	Exception
`Value`	`public Array Value {get; set;}`	Get and set the parameter value.	`xPCException` — When problem occurs, query xPCException object `Reason` property.

Introduced in R2011b

xPCParameters Class

Access run-time parameters

Syntax

```
public class xPCParameters : xPCApplicationObject
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Methods

Method	Description
`xPCParameters.-LoadParameterSet`	Load parameter values for real-time application
`xPCParameters.Refresh`	Refresh state of object
`xPCParameters.-SaveParameterSet`	Save parameter values of real-time application

Properties

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

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

Introduced in R2011b

xPCScope Class

Access Simulink Real-Time scopes

Syntax

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

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Methods

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

Events

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

Properties

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

Property	C# Declaration Syntax	Description	Exception
`NumSamples`	`public int NumSamples {get; set;}`	Get or set number of contiguous samples captured during the acquisition of a data package. 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 is possible that your data contains zeroes.	`xPCException` — When problem occurs, query xPCException object `Reason` property.
`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.

Property	C# Declaration Syntax	Description	Exception
`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'`, `TriggerLevel`indicates the value the signal has to cross to trigger the scope and start acquiring data. You can cross the trigger level with either a rising or falling signal.	`xPCException` — When problem occurs, query xPCException object `Reason` property.
`TriggerMode`	`public SCTRIGGERMODE TriggerMode {get; set;}`	Get or set trigger mode for a scope. Valid values are `'FreeRun'` (default), `'Software'`, `'Signal'`, and `'Scope'`.	`xPCException` — When problem occurs, query xPCException object `Reason` property.
`TriggerScope`	`public int TriggerScope {get; set;}`	If `TriggerMode` is `'Scope'`, `TriggerScope` 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.

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

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

- Autorestart is on — When the scope triggers, the scope starts collecting 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 starts collecting data into a memory buffer. It stops when it has collected the number of samples that you

specified. A background task examines the buffer and writes data to the disk continuously, appending the new data to the end of the file.

Introduced in R2011b

xPCScopeCollectionEventArgs Class

`xPCScopeCollection.Added` event data

Syntax

```
public class xPCScopeCollectionEventArgs : EventArgs
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Properties

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

Introduced in R2011b

xPCScopeRemCollectionEventArgs Class

`xPCScopeCollection.Removed` event data

Syntax

```
public class xPCScopeRemCollectionEventArgs : EventArgs
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Properties

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

Introduced in R2011b

xPCScopeSignalCollectionEventArgs Class

`xPCScopeSignalCollection.Added` event data

Syntax

```
public class xPCScopeSignalCollectionEventArgs : EventArgs
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Properties

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

Introduced in R2011b

xPCScopes Class

Access scope objects

Syntax

```
public class xPCScopes : xPCApplicationObject
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Methods

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

Properties

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

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

Introduced in R2011b

xPCSignal Class

Access signal objects

Syntax

```
public class xPCSignal : xPCApplicationObject
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Methods

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

Properties

Property	C# Declaration Syntax	Description
`BlockPath`	`public virtual string BlockPath {get;}`	Get block path name (signal name) of the signal.
`DataType`	`public virtual string DataType {get;}`	Get Simulink data type name.

Property	C# Declaration Syntax	Description
Label	`public virtual string Label {get;}`	Get label of signal. If no label is associated with the signal, this property returns an empty character string.
SignalId	`public virtual int SignalId {get;}`	Get numeric identifier that represents the signal object.
UserData	`public Object UserData {get; set;}`	Get and set user-defined object that you can use to store and retrieve additional information.
Width	`public virtual int Width {get;}`	Get signal width.

Introduced in R2011b

xPCSignals Class

Access signal objects

Syntax

```
public class xPCSignals : xPCApplicationObject
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Methods

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

Properties

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

Property	C# Declaration Syntax	Description	Exception

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

Introduced in R2011b

xPCStateLogger Class

Access to state log

Syntax

```
public class xPCStateLogger : xPCLog
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Properties

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

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

Introduced in R2011b

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

Method	Description
xPCTargetPC.DisconnectAsync	Asynchronous request to disconnect from target computer
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

Event	Description
xPCTargetPC.Loaded	Event after xPCTargetPC.Load is complete
xPCTargetPC.Loading	Event before xPCTargetPC.Load starts
xPCTargetPC.-RebootCompleted	Event when xPCTargetPC.RebootAsync is complete
xPCTargetPC.Rebooted	Event after xPCTargetPC.Reboot is complete
xPCTargetPC.Rebooting	Event before xPCTargetPC.Reboot starts
xPCTargetPC.-UnloadCompleted	Event when xPCTargetPC.UnloadAsync is complete
xPCTargetPC.Unloaded	Event after xPCTargetPC.Unload is complete
xPCTargetPC.Unloading	Event before xPCTargetPC.Unload starts

Properties

Property	C# Declaration Syntax	Description	Exception
AppFileName	public string AppFileName {get; set;}	Get or set the full path name to the real-time application, without file extension.	
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.

Property	C# Declaration Syntax	Description	Exception
`Component`	`public IComponent Component {get;}`	Get component associated with the ISite when implemented by a class.	
`Container`	`public IContainer Container {get;}`	Get the IContainer associated with the ISite when implemented by a class.	
`Container-Control`	`public ContainerControl ContainerControl {get; set;}`	Provide focus-management functionality for controls that can function as containers for other controls.	
`DLMFileName`	`public string DLMFileName {get; set;}`	Get or set the full path to the DLM file name. **Note** `AppFileName` has superseded this property.	
`Echo`	`public bool Echo {get; set;}`	Get or set the target display on the target computer.	`xPCException` — When problem occurs, query xPCException object `Reason` property.
`FileSystem`	`public xPCFileSystem FileSystem {get;}`	Get a reference to an `xPCFileSystem` object that you can use to interface with the target file system. If no communication is established, the property returns a NULL object.	

Property	C# Declaration Syntax	Description	Exception
`HostTargetComm`	`public XPCProtocol HostTargetComm {get; set;}`	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`	`public bool IsConnected {get;}`	Get connection status (established or not) to a remote target computer.	
`IsConnecting-Busy`	`public bool IsConnectingBusy {get;}`	Get `ConnectAsync` request status (in progress or not).	
`IsDiscon-nectingBusy`	`public bool IsDisconnectingBusy {get;}`	Get whether a `DisconnectAsync` request is in progress.	
`IsLoadingBusy`	`public bool IsLoadingBusy {get;}`	Gets `LoadAsync` request status (in progress or not).	
`IsRebooting-Busy`	`public bool IsRebootingBusy {get;}`	Get `RebootAsync` request status (in progress or not).	
`IsUnloading-Busy`	`public bool IsUnloadingBusy {get;}`	Gets `unLoadingAsync` request status (in progress or not).	
`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.	

Property	C# Declaration Syntax	Description	Exception
`TargetPCName`	`public string TargetPCName {get; set;}`	Get or set a value indicating the target computer name associated with the target computer.	
`TcpIpTarget-Address`	`public string TcpIpTargetAddress {get; set;}`	Get or set a valid IP address for your target computer.	
`TcpIpTarget-Port`	`public string TcpIpTargetPort {get; set;}`	Get or set the TCP/IP target port. The default is `22222`. 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.	

Introduced in R2011b

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

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

Introduced in R2011b

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

Introduced in R2011b

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 Item[string blkpath] {get;}`	Get xPCTgtScopeSignal object from signal name (*`blkpath`*). *`blkpath`* is the signal name that represents a signal object added to its parent xPCTargetScope object. This property returns the file scope signal object as type xPCTgtScopeSignal.	`xPCException` — When problem occurs, query xPCException object `Reason` property.

Introduced in R2011b

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.

Introduced in R2011b

xPCTgtScopeSignal Class

Access to target scope signals

Syntax

```
public class xPCTgtScopeSignal : xPCScopeSignal
```

Description

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Properties

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

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

xPCTargetPC.Ping

Test communication between development and target computers

Syntax

```
public bool Ping()
public string Ping('info')
public string Ping('reset')
```

Description

Class: `xPCTargetPC Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public bool Ping()` tests at a low level the connection between the development and target computers. This method returns a Boolean value. If a data channel is open between the development and target computers, `Ping()` leaves it open.

`public string Ping('info')` returns human-readable information about the connection between the development and target computers. If a data channel is open between the development and target computers, `Ping('info')` leaves it open.

If the target computer is connected, `Ping('info')` uses the information/control channel to return a string of the form:

`'xPCTarget`*`version hex_address`*` Connected'`

In this string:

- *`version`* — Version of the Simulink Real-Time kernel that is running on the target computer. For example, the function returns `xPCTarget6.6` for kernel version `6.6`.

- *hex_address* — Hexadecimal representation of the development computer network address to which the target computer is connected. The hexadecimal digits are reversed from the digits of the network address. For example, `0x640a0a0a` represents the network address `10.10.10.100`.

  When the target computer is not connected to a development computer, *hex_address* is a random hexadecimal number.

If the target computer is not connected to a development computer, `Ping('info')` returns a string of the form:

```
'xPCTarget version hex_address Disconnected'
```

`public string Ping('reset')` uses the information/control channel to close an open communication channel between the development and target computers and returns a string of the form:

```
'xPCTarget version hex_address Disconnected'
```

Introduced in R2011b

xPCFileStream.Read

Read block of bytes from stream and write data to buffer

Syntax

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

Description

Class: `xPCFileStream Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

`public long Read(byte[] buffer, long offset, long 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. If the requested number of bytes are not currently available, *`count`* is less than the number of bytes requested. If the method reaches the end of the stream, it can also be zero.

The largest single file that you can create on the target computer is 4 GB.

Exception

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

Introduced in R2011b

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.

Introduced in R2011b

xPCTargetPC.RebootAsync

Asynchronous request to restart target computer

Syntax

```
public void RebootAsync()
```

Description

Class: `xPCTargetPC Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Exception

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

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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().

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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().

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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 in the operating system on the target computer can have a maximum of 260 characters. If the file name is longer than eight-dot-three format (eight character file name, period, three character extension), the operating system represents the file name in truncated form (for example, six characters followed by `'~1'`). MATLAB commands can access the file using the fully qualified file name or the truncated representation of the name. Some block parameters, such as the Scope block `filename` parameter, require 8.3 format for the file name.

Exception

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

Introduced in R2011b

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.

Introduced in R2011b

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.
`ROLLING`	Specifies target scope drawing mode to rolling mode.
`SLIDING`	The value `SLIDING` will be removed in a future release. It behaves like value `ROLLING`.

Introduced in R2009b

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.

Introduced in R2009b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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`	You have stopped (interrupted) the scope.
`PREACQUIRING`	Scope acquires a predefined number of samples before triggering.

Introduced in R2009b

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`	Another scope triggers this scope at a predefined trigger point of the triggering scope. You modify this trigger point with the value of `TriggerScopeSample`.

Introduced in R2009b

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.

Introduced in R2009b

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.

Introduced in R2009b

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.

Introduced in R2011b

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.

Introduced in R2011b

xPCParameter.SetParamAsync

Asynchronous request to change parameter value on target computer

Syntax

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

Description

Class: `xPCParameter Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

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

Exception

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

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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 of the file scopes in the xPCFileScopeCollection.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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 allof the file scopes in the xPCFileScopeCollection.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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. If a data channel is open between the development and target computers, this method leaves it open.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

xPCTargetPC.UnloadAsync

Asynchronous request to unload real-time application from target computer

Syntax

```
public void UnloadAsync()
```

Description

Class: `xPCTargetPC Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Exception

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

Introduced in R2011b

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.

Introduced in R2011b

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.

Introduced in R2011b

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 starts to unload from a target computer.

Introduced in R2011b

xPCFileStream.Write

Write block of bytes to file stream

Syntax

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

Description

Class: `xPCFileStream Class`

Method

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

Exception

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

Introduced in R2011b

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.

Introduced in R2011b

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.
`Starting`	Real-time application is starting.
`Running`	Real-time application is running.

Introduced in R2009b

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.

Introduced in R2009b

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.

Introduced in R2009b

xPCException

Construct new instance of `xPCException` class

Syntax

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

Description

Class: `xPCException Class`

Constructor

Namespace: `MathWorks.xPCTarget.FrameWork`

Syntax Language: C#

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

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

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

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

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

Introduced in R2009b

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.

Introduced in R2009b

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 at most 128 files open on the target computer at the same time.
- The largest single file that you can create on the target computer 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 in the operating system on the target computer can have a maximum of 260 characters. If the file name is longer than eight-dot-three format (eight character file name, period, three character extension), the operating system represents the file name in truncated form (for example, six characters followed by `'~1'`). MATLAB commands can access the file using the fully qualified file name or the truncated representation of the name. Some block parameters, such as the Scope block `filename` parameter, require 8.3 format for the file name.

- Do not write data to the `private` folder on your target computer. It is reserved for Simulink Real-Time internal use.

Exception

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

Introduced in R2009b

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

Introduced in R2009b

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 at most 128 files open on the target computer at the same time.
- The largest single file that you can create on the target computer 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 in the operating system on the target computer can have a maximum of 260 characters. If the file name is longer than eight-dot-three format (eight character file name, period, three character extension), the operating system represents the file name in truncated form (for example, six characters followed by

'~1'). MATLAB commands can access the file using the fully qualified file name or the truncated representation of the name. Some block parameters, such as the Scope block `filename` parameter, require 8.3 format for the file name.

- Do not write data to the `private` folder on your target computer. It is reserved for Simulink Real-Time internal use.

Exception

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

Introduced in R2009b

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.

Introduced in R2009b

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

Introduced in R2009b

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

Introduced in R2009b

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

Introduced in R2009b

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.

Introduced in R2009b

4

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.
- You can call the API functions from non-C languages, such as C++ and Java. Refer to the compiler documentation of the non-C language for a description of how to access C functions from a library DLL. To access the Simulink Real-Time C API DLL, follow these directions.
- You can work with real-time applications with either MATLAB or a Simulink Real-Time C API control application. However, only one control 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 command frees the connection to the target computer for use by your Simulink Real-Time C API application. Conversely, to access the target from a MATLAB session, you must quit your control application, or do the equivalent of calling the function `xPCClosePort`.
- 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, they exit with the global variable `xPCError` set to `ETCPTIMEOUT`. Use the `xPCGetLoadTimeOut` and `xPCSetLoadTimeOut` functions to get and set the timeout values, respectively.

A few things that are common to almost all the functions in the Simulink Real-Time C API are not covered in the reference topics for the individual functions.

- Almost every function (except `xPCOpenTcpIpPort`, `xPCGetLastError`, and `xPCErrorMsg`) has as one of its parameters the integer variable *`port`*. `xPCOpenTcpIpPort` returns this variable to represent the communications link with the target computer.
- Almost every function (except `xPCGetLastError` and `xPCErrorMsg`) sets a global error value when an error occurs. The application obtains this value by calling the function `xPCGetLastError`, and retrieves a descriptive character string about the error by using the function `xPCErrorMsg`. The actual error values are subject to change. However, a zero value typically means that the operation completed without

producing an error, while a nonzero value typically signifies an error condition. The library resets the error value every time an API function is called; therefore, 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`.

5

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

Name	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.
Ext	This value contains the file type of the element, if the element is a file (*isDir* is 0). If the element is a folder (*isDir* is 1), this field is empty.
Day	This value contains the day the file or folder was last modified.
Month	This value contains the month the file or folder was last modified.
Year	This value contains the year the file or folder was last modified.
Hour	This value contains the hour the file or folder was last modified.

Min	This value contains the minute the file or folder was last modified.
isDir	This value indicates if the element is a file (`0`) or folder (`1`). If it is a folder, `Bytes` has a value of `0`.
Size	This value contains the size of the file in bytes. If the element is a folder, this value is `0`.

Description

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

See Also

`xPCFSDirItems`

Introduced in R2007a

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;
   unsigned int DriveType;
} diskinfo;
```

Fields

Label	This value contains the zero-terminated character string that contains the volume label. The character string is empty if the volume has no label.
DriveLetter	This value contains the drive letter, in uppercase.
Reserved	Reserved.
SerialNumber	This value contains the volume serial number.

FirstPhysicalSector	This value contains the logical block addressing (LBA) address of the logical drive boot record. For 3.5-inch disks, this value is `0`.
FATType	This value contains the type of file system found. It contains `32`, representing FAT-32 volumes. The values `12` and `16`, representing FAT-12 and FAT-16 volumes, are supported for backward compatibility only.
FATCount	This value contains the number of FAT partitions on the volume.
MaxDirEntries	This value contains the size of the root folder. For FAT-32 systems, this value is `0`.
BytesPerSector	This value contains the sector size. This value is most likely to be `512`.
SectorsPerCluster	This value contains, in sectors, the size of the smallest unit of storage that can be allocated to a file.
TotalClusters	This value contains the number of file storage clusters on the volume.
BadClusters	This value contains the number of clusters that have been marked as bad. These clusters are unavailable for file storage.
FreeClusters	This value contains the number of clusters that are currently available for storage.
Files	This value contains the number of files, including folders, on the volume. This number excludes the root folder and files that have an allocated file size of 0.
FileChains	This value contains the number of contiguous cluster chains. On a defragmented volume, this value is identical to the value of `Files`.
FreeChains	This value contains the number of contiguous cluster chains of free clusters. On a defragmented volume, this value is `1`.
LargestFreeChain	This value contains the maximum allocated file size, in number of clusters, for a newly allocated contiguous file. On a defragmented volume, this value is identical to `FreeClusters`.

DriveType

This value contains a code for the type of permanent storage installed in the target computer. The values are:

- `0` — Unknown drive
- `1` — Drive with no root folder
- `2` — Removable drive
- `3` — Fixed (hard) drive
- `4` — Remote drive (not supported)
- `5` — CDROM drive (not supported)
- `6` — RAM disk

Description

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

See Also

`xPCFSDiskInfo`

Introduced in R2012a

fileinfo

Type definition for file information structure

Syntax

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

Fields

FilePos	This value contains the current file pointer.
AllocatedSize	This value contains the currently allocated file size.
ClusterChains	This value indicates how many separate cluster chains are allocated for the file.
VolumeSerialNumber	This value holds the serial number of the volume the file resides on.
FullName	This value contains a copy of the complete path name of the file. This field is valid only while the file is open.

Description

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

There are the following limitations:

- You can have at most 128 files open on the target computer at the same time.
- The largest single file that you can create on the target computer 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 in the operating system on the target computer can have a maximum of 260 characters. If the file name is longer than eight-dot-three format (eight character file name, period, three character extension), the operating system represents the file name in truncated form (for example, six characters followed by `'~1'`). MATLAB commands can access the file using the fully qualified file name or the truncated representation of the name. Some block parameters, such as the Scope block `filename` parameter, require 8.3 format for the file name.
- Do not write data to the `private` folder on your target computer. It is reserved for Simulink Real-Time internal use.

See Also

`xPCFSFileInfo`

Introduced before R2006a

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

Introduced before R2006a

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

number	The scope number.
type	Determines whether the scope is displayed on the development computer or on the target computer. Values are one of the following: 1 Host 2 Target
state	Indicates the scope state. Values are one of the following: 0 Waiting to start 1 Scope is waiting for a trigger 2 Data is being acquired

	3 Acquisition is finished 4 Scope is stopped (interrupted) 5 Scope is preacquiring data
signals	List of signal indices from the target object to display on the scope. Target scopes are restricted to `10` signals.
numsamples	Number of contiguous samples captured during the acquisition of a data package.
decimation	If `1`, acquire every sample in a scope window. Otherwise, acquire every *n*th sample in a scope window.
triggermode	Trigger mode for a scope. Values are one of the following: `0` FreeRun (default) `1` Software `2` Signal `3` Scope
numprepostsamples	If this value is less than `0`, *numprepostsamples* is the number of samples to be saved before a trigger event. If this value is greater than `0`, *numprepostsamples* is the number of samples to skip after the trigger event before data acquisition begins.
triggersignal	If *triggermode* is `2` (Signal), *triggersignal* identifies the block output signal to use for triggering the scope. Identify the signal with a signal index.
triggerscope	If *triggermode* is `3` (Scope), *triggerscope* identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered.
triggerscopesample	If *triggermode* is `3` (Scope), *triggerscopesample* specifies the number of samples to be acquired by the triggering scope before triggering a second scope. This value must be nonnegative.
triggerlevel	If *triggermode* is `2` (Signal), *triggerlevel* indicates the value the signal has to cross to trigger the scope to start acquiring data. The trigger level can be crossed with either a rising or falling signal.

triggerslope	If *triggermode* is 2 (Signal), indicates whether the trigger is on a rising or falling signal. Values are: 0 Either rising or falling (default) 1 Rising 2 Falling

Description

The scopedata structure holds the data about a scope used in the functions xPCGetScope and xPCSetScope. In the structure, the fields are as in the various xPCGetSc* functions. For example, *state* is as in xPCScGetState, *signals* is as in xPCScGetSignals. 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

Introduced before R2006a

xPCAddScope

Create scope

Prototype

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

Arguments

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

Description

The xPCAddScope function creates a 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 | SimulinkRealTime.target

Introduced before R2006a

xPCAverageTET

Return average task execution time

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.

Return

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 | `SimulinkRealTime.target`

Introduced before R2006a

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. A subsequent call to xPCOpenConnection succeeds without the need to resupply communication data such as the IP address or port number. To close the communication channel completely, 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

Introduced before R2006a

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 | `SimulinkRealTime.target`

Introduced before R2006a

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. Use this function to end 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

Introduced before R2006a

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

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

Introduced before R2006a

xPCFreeAPI

Unload Simulink Real-Time DLL

Prototype

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

Description

The `xPCFreeAPI` function unloads the Simulink Real-Time dynamic link library. To unload the Simulink Real-Time API DLL and free the memory allocated to the API functions, call this function once at the end of your custom program. 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

Introduced before R2006a

xPCFSCD

Change current folder on target computer to specified path

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
dir	Enter the path to the new folder on the target computer.

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 | SimulinkRealTime.fileSystem

Introduced before R2006a

xPCFSCloseFile

Close file on target computer

Prototype

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

Arguments

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

Description

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

See Also

xPCFSOpenFile | xPCFSReadFile | xPCFSWriteFile | `SimulinkRealTime.fileSystem`

Introduced before R2006a

xPCFSDir

Get contents of specified folder on target computer

Prototype

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

Arguments

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

Description

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

See Also

xPCFSDirSize | `SimulinkRealTime.fileSystem`

Introduced before R2006a

xPCFSDirItems

Get contents of specified folder on target computer

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
path	Enter the path on the target computer.
dirs	Enter the structure for receiving the contents of the folder.
numDirItems	Enter the number of items in the folder.

Description

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

See Also

`SimulinkRealTime.fileSystem` | `dirStruct` | `xPCFSDirStructSize`

Introduced in R2007a

xPCFSDirSize

Return size of specified folder listing on target computer

Prototype

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

Arguments

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

Return

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

`SimulinkRealTime.fileSystem` | `xPCFSDirItems`

Introduced before R2006a

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

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 | `SimulinkRealTime.fileSystem`

Introduced in R2007a

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`

Introduced in R2006a

xPCFSFileInfo

Return information for open file on target computer

Prototype

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

Arguments

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

Description

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

See Also

`File System`

Introduced in R2008b

xPCFSGetError

Get text description for error number on target computer file system

Prototype

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

Arguments

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

Description

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

Introduced before R2006a

xPCFSGetFileSize

Return size of file on target computer

Prototype

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

Arguments

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

Return

Returns 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.

The largest single file that you can create on the target computer is 4 GB.

See Also

xPCFSOpenFile | xPCFSReadFile | `SimulinkRealTime.fileSystem`

Introduced before R2006a

xPCFSGetPWD

Get current folder of target computer

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
pwd	The path of the current folder is stored in *pwd*.

Description

The `xPCFSGetPWD` function places the path of the current folder on the target computer in *pwd*. The caller must allocate an array and pass it into *pwd*.

See Also

`File System`

Introduced before R2006a

xPCFSMKDIR

Create 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 new folder 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 | `SimulinkRealTime.fileSystem`

Introduced before R2006a

xPCFSOpenFile

Open file on target computer

Prototype

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

Arguments

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

Return

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 at most 128 files open on the target computer at the same time.
- The largest single file that you can create on the target computer 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 in the operating system on the target computer can have a maximum of 260 characters. If the file name is longer than eight-dot-three format (eight character file name, period, three character extension), the operating system represents the file name in truncated form (for example, six characters followed by `'~1'`). MATLAB commands can access the file using the fully qualified file name or the truncated representation of the name. Some block parameters, such as the Scope block `filename` parameter, require 8.3 format for the file name.
- Do not write data to the `private` folder on your target computer. It is reserved for Simulink Real-Time internal use.

See Also

xPCFSCloseFile | xPCFSGetFileSize | xPCFSReadFile | xPCFSWriteFile | `SimulinkRealTime.fileSystem`

Introduced before R2006a

xPCFSReadFile

Read open file on target computer

Prototype

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

Arguments

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

Description

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

The largest single file that you can create on the target computer is 4 GB.

See Also

xPCFSCloseFile | xPCFSGetFileSize | xPCFSOpenFile | xPCFSWriteFile | SimulinkRealTime.fileSystem

Introduced before R2006a

xPCFSRemoveFile

Remove file from target computer

Prototype

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

Arguments

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

Description

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

See Also

File System

Introduced before R2006a

xPCFSRMDIR

Remove folder from target computer

Prototype

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

Arguments

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

Description

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

See Also

File System

Introduced before R2006a

xPCFSScGetFilename

Get name of file for scope

Prototype

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

Arguments

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

Return

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

Description

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

See Also

xPCFSScSetFilename | Real-Time File Scope

Introduced before R2006a

xPCFSScGetWriteMode

Get write mode of file for scope

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
scNum	Enter the scope number.

Return

Returns the number indicating the write mode. Values are

`0`	Lazy mode. The FAT entry is updated only when the file is closed and not during each file write operation. This mode is faster than commit mode. However, if the system crashes before the file is closed, the file system does not have the actual file size. (The file contents, however, are intact.)
`1`	Commit mode. Each file write operation simultaneously updates the FAT entry for the file. This mode is slower than lazy mode, but the file system maintains the actual file size.

Description

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

See Also

xPCFSScSetWriteMode | `Real-Time File Scope`

Introduced before R2006a

xPCFSScGetWriteSize

Get block write size of data chunks

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
scNum	Enter the scope number.

Return

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

Description

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

See Also

xPCFSScSetWriteSize | `Real-Time File Scope`

Introduced before R2006a

xPCFSScSetFilename

Specify name for file to contain signal data

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
scNum	Enter the scope number.
filename	Enter the name of a file for receiving the signal data.

Description

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

A fully qualified file name in the operating system on the target computer can have a maximum of 260 characters. If the file name is longer than eight-dot-three format (eight character file name, period, three character extension), the operating system represents the file name in truncated form (for example, six characters followed by `'~1'`). MATLAB commands can access the file using the fully qualified file name or the truncated representation of the name. Some block parameters, such as the Scope block `filename` parameter, require 8.3 format for the file name.

See Also

xPCFSScGetFilename | `Real-Time File Scope`

Introduced before R2006a

xPCFSScSetWriteMode

Specify when file allocation table entry is updated

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
scNum	Enter the scope number.
writeMode	Enter an integer for the write mode: `0` Enables lazy write mode `1` Enables commit write mode

Description

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

`0`	Lazy mode. The FAT entry is updated only when the file is closed and not during each file write operation. This mode is faster than commit mode. However, if the system crashes before the file is closed, the file system does not have the actual file size. (The file contents, however, are intact.)
`1`	Commit mode. Each file write operation simultaneously updates the FAT entry for the file. This mode is slower than lazy mode, but the file system maintains the actual file size.

See Also

xPCFSScGetWriteMode | `Real-Time File Scope`

Introduced before R2006a

xPCFSScSetWriteSize

Specify that memory buffer collect data in multiples of write size

Prototype

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

Arguments

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

Description

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

See Also

xPCFSScGetWriteSize | `Real-Time File Scope`

Introduced before R2006a

xPCFSWriteFile

Write to file on target computer

Prototype

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

Arguments

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

Description

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

See Also

xPCFSCloseFile | xPCFSGetFileSize | xPCFSOpenFile | xPCFSReadFile

Introduced before R2006a

xPCGetAPIVersion

Get version number of Simulink Real-Time API

Prototype

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

Return

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`

Introduced in R2007a

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. 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

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. If an error occurs, 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 | `SimulinkRealTime.target`

Introduced before R2006a

xPCGetEcho

Return display mode for target message window

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.

Return

Returns a 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.
-1	The function detected an error.

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

Introduced before R2006a

xPCGetExecTime

Return real-time application execution time

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.

Return

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 | `SimulinkRealTime.target`

Introduced before R2006a

xPCGetLastError

Return constant of last error

Prototype

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

Return

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, 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

Introduced before R2006a

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

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

Description

The `xPCGetLoadTimeOut` function returns the number of seconds allowed for communication between the development computer and the real-time application running on the target computer. When a Simulink Real-Time API function initiates communication, it waits for some seconds before checking if the communication is complete. If 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

Topics

“Troubleshoot Communication Timeout with Target Computers”

Introduced before R2006a

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

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

Introduced before R2006a

xPCGetNumOutputs

Return number of outputs

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.

Return

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

Introduced before R2006a

xPCGetNumParams

Return number of tunable parameters

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.

Return

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 | `SimulinkRealTime.target`

Introduced before R2006a

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

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.

Introduced in R2008b

xPCGetNumSignals

Return number of signals

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.

Return

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 | `SimulinkRealTime.target`

Introduced before R2006a

xPCGetNumStates

Return number of states

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.

Return

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 | `SimulinkRealTime.target`

Introduced before R2006a

xPCGetOutputLog

Copy output log data to array

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
first_sample	Enter the index of the first sample to copy.
num_samples	Enter the number of samples that the function is to copy from the output log.
decimation	if 1, copy every sample value. Otherwise, copy every Nth sample value.
output_id	Enter an output identification number.
output_data	The log is stored in *output_data*, whose allocation is the responsibility of the caller.

Description

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

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

See Also

xPCNumLogWraps | xPCNumLogSamples | xPCMaxLogSamples | xPCGetNumOutputs | xPCGetStateLog | xPCGetTETLog | xPCGetTimeLog | `SimulinkRealTime.target`

Introduced before R2006a

xPCGetParam

Get parameter value and copy it to array

Prototype

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

Arguments

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

Description

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

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

See Also

xPCSetParam | xPCGetParamDims | xPCGetParamIdx | xPCGetNumParams | `SimulinkRealTime.target`

Introduced before R2006a

xPCGetParamDims

Get row and column dimensions of parameter

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
paramIndex	Parameter index.
dimension	Dimensions (row, column) of a parameter.

Description

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

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

See Also

xPCGetParam | xPCGetParamName | xPCGetParamDims | xPCGetParamIdx | xPCGetNumParams | xPCSetParam | `SimulinkRealTime.target`

Introduced before R2006a

xPCGetParamIdx

Return parameter index

Prototype

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

Arguments

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

Return

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 the names that were generated at real-time application building time. To find the block names, access the file *model_name*`pt.m` in the generated code, where *model_name* is the name of the model. A block can have one or more parameters.

See Also

xPCGetParam | xPCGetParamName | xPCGetParamDims | `SimulinkRealTime.target`

Introduced before R2006a

xPCGetParamName

Get name of parameter

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
paramIdx	Enter a parameter index.
blockName	Character string with the full block path generated by Simulink Coder.
paramName	Name of a parameter for a specific block.

Description

The `xPCGetParamName` function gets the parameter name and block name for a parameter with the index *paramIdx*. The block path and name are returned and stored in *blockName*, and the parameter name is returned and stored in *paramName*. 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 | `SimulinkRealTime.target`

Introduced before R2006a

xPCGetSampleTime

Return real-time application sample time

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.

Return

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 | `SimulinkRealTime.target`

Introduced before R2006a

xPCGetScope

Get and copy scope data to structure

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
scNum	Enter the scope number.

Return

Returns a structure of type `scopedata`.

Description

Note The `xPCGetScope` function will be removed in a future release. Use the `xPCScGet`*`ScopePropertyName`* functions to access property values instead. For example, to get the number of samples being acquired in one data acquisition cycle, use `xPCScGetNumSamples`.

The `xPCGetScope` function gets properties of a scope with *`scNum`* and copies the properties into a structure with type `scopedata`. You can use this function 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 | `SimulinkRealTime.target`

Introduced before R2006a

xPCGetScopeList

Get and copy list of scope numbers

Prototype

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

Arguments

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

Description

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

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

Introduced in R2008b

xPCGetScopes

Get and copy list of scope numbers

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
data	List of scope numbers in an integer array (allocated by the caller) as a list of unsorted integers and terminated by `-1`.

Description

The `xPCGetScopes` function gets the list of scopes currently defined. You can use the constant `MAX_SCOPES` (defined in `xpcapiconst.h`) as the size of *`data`*. `MAX_SCOPES` is set to `30`.

Note This function will be removed in a future release. Use the `xPCGetScopeList` function instead.

See Also

xPCSetScope | xPCGetScope | xPCScGetSignals | `SimulinkRealTime.target`

Introduced before R2006a

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

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.

Introduced in R2008b

xPCGetSignal

Return value of signal

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
sigNum	Enter a signal number.

Return

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 | `SimulinkRealTime.target`

Introduced before R2006a

xPCGetSignalIdx

Return index for signal

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
sigName	Enter a signal name.

Return

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. To find the name, access the file *model_name*`bio.m` in the generated code, where *model_name* is the name of the model. The creator of the custom program already knows the signal name.

See Also

xPCGetSignalName | xPCGetSignalWidth | xPCGetSignal | xPCGetSignals | `SimulinkRealTime.target`

Introduced before R2006a

xPCGetSigIdxfromLabel

Return array of signal indices

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
sigLabel	Character string with the name of a signal label.
sigIds	Return array of signal indices.

Return

If the function finds a signal, it fills an array *sigIds* with signal indices and returns `0`. If it does not find a 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” (Simulink)). The Simulink Real-Time software refers to Simulink signal names as signal labels. The creator of the custom program already knows 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 amount of memory that the program must allocate for the `sigIds` array.

See Also

xPCGetSigLabelWidth | xPCGetSignalLabel

Introduced in R2007a

xPCGetSignalLabel

Copy label of signal to character array

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
sigIdx	Enter signal index.
sigLabel	Return signal label associated with signal index, *sigIdx*.

Return

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” (Simulink)). The Simulink Real-Time software refers to Simulink signal names as signal labels. The creator of the custom program already knows the signal name/label.

See Also

`xPCGetSigIdxfromLabel` | `xPCGetSigLabelWidth`

Introduced in R2007a

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

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” (Simulink)). The Simulink Real-Time software refers to Simulink signal names as signal labels. The creator of the custom program already knows the signal name/label. Signal labels must be unique.

See Also

`xPCGetSigIdxfromLabel` | `xPCGetSignalLabel`

Introduced in R2007a

xPCGetSignalName

Copy name of signal to character array

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
sigIdx	Enter a signal index.
sigName	Character string with the name of a signal.

Return

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 | `SimulinkRealTime.target`

Introduced before R2006a

xPCGetSignals

Return vector of signal values

Prototype

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

Arguments

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

Return

If the function completes execution without detecting an error, it returns `0`. 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 function acquires some signal values in one time step and later signals in another. To acquire signal values within one time step, 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`*, store the list you provide 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 get the signals repeatedly, repeat the call to `xPCGetSignals`. If you do not change `sigID`, call xPCGetSignalIdx only once.

See Also

xPCGetSignal | xPCGetSignalIdx

Introduced before R2006a

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

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 is useful to reassemble the components into a vector again. The width of a signal is the number of signals in the vector.

See Also

xPCGetSignalIdx | xPCGetSignalName | xPCGetSignal | xPCGetSignals

Introduced before R2006a

xPCGetStateLog

Copy state log values to array

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
first_sample	Enter the index of the first sample to copy.
num_samples	Enter the number of samples that the function is to copy from the output log.
decimation	Select whether to copy all the sample values or every N th value.
state_id	Enter a state identification number.
state_data	The log is stored in *state_data*, whose allocation is the responsibility of the caller.

Description

The xPCGetStateLog function gets the state log. It then copies the log into *state_data*. You get the data for each state signal in turn by specifying the *state_id*. State IDs range from 1 to (N-1), where N is the return value of xPCGetNumStates. Entering 1 for *decimation* copies all values. Entering N copies every N th value. For *first_sample*, the sample indices range from 0 to (N-1), where N is the return value of xPCNumLogSamples. Use the xPCNumLogSamples function to get the maximum number of samples.

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

See Also

xPCNumLogWraps | xPCNumLogSamples | xPCMaxLogSamples | xPCGetNumStates | xPCGetOutputLog | xPCGetTETLog | xPCGetTimeLog | `SimulinkRealTime.target`

Introduced before R2006a

xPCGetStopTime

Return stop time

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.

Return

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 amount of time, in seconds, that the real-time application runs before stopping. If the function detects an error, it returns `-10.0`. Use the function xPCGetLastError to find the error number.

See Also

xPCSetStopTime | `SimulinkRealTime.target`

Introduced before R2006a

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`

Introduced in R2007a

xPCGetTETLog

Copy TET log to array

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
first_sample	Enter the index of the first sample to copy.
num_samples	Enter the number of samples that the function is to copy from the TET log.
decimation	Select whether to copy all the sample values or every Nth value.
TET_data	The log is stored in *TET_data*, whose allocation is the responsibility of the caller.

Description

The xPCGetTETLog function gets the task execution time (TET) log. It then copies the log into *TET_data*. Entering 1 for *decimation* copies all values. Entering N copies every 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.

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

See Also

xPCNumLogWraps | xPCNumLogSamples | xPCMaxLogSamples | xPCGetNumOutputs | xPCGetStateLog | xPCGetTimeLog | `SimulinkRealTime.target` | `SimulinkRealTime.utils.TETMonitor.open`

Introduced before R2006a

xPCGetTimeLog

Copy time log to array

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
first_sample	Enter the index of the first sample to copy.
num_samples	Enter the number of samples that the function is to copy from the time log.
decimation	Select whether to copy all the sample values or every Nth value.
time_data	The log is stored in *time_data*, whose allocation is the responsibility of the caller.

Description

The xPCGetTimeLog function gets the time log and copies the log into *time_data*. This function is especially useful in the case of value-equidistant logging, where the logged values are not necessarily spaced uniformly 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.

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

See Also

xPCGetLogMode | xPCSetLogMode | xPCGetTETLog | xPCGetStateLog | xPCMaxLogSamples | xPCNumLogSamples | xPCNumLogWraps | `SimulinkRealTime.target`

Introduced before R2006a

xPCInitAPI

Initialize Simulink Real-Time DLL

Prototype

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

Return

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

Description

The `xPCInitAPI` function initializes the Simulink Real-Time dynamic link library. To load the Simulink Real-Time API DLL, execute `xPCInitAPI` once at the beginning of the custom program. 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

Introduced before R2006a

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 | `SimulinkRealTime.target`

Introduced before R2006a

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 | `SimulinkRealTime.target`

Introduced before R2006a

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 function returns `1`. If the scope is in the process of acquiring data, it 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. Use the xPCGetScope function to get the scope number.

You can call this function for target scopes; however, because target scopes restart immediately, it is almost impossible to find them in the finished state.

See Also

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

Introduced before R2006a

xPCLoadApp

Load real-time application onto target computer

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
pathstr	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"`.
filename	Enter the name of a compiled real-time application 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. If the real-time application is in the current folder, you can set *`pathstr`* to NULL or to the character string `'nopath'`. The variable *`filename`* must not contain the real-time application extension.

See Also

xPCGetLoadTimeOut | xPCSetLoadTimeOut | xPCUnloadApp | xPCStopApp | xPCStartApp | `SimulinkRealTime.target`

Introduced before R2006a

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 on a local drive of the target computer. The parameter file must have been saved from a previous call to xPCSaveParamSet.

See Also

xPCSaveParamSet

Introduced before R2006a

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

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 | `SimulinkRealTime.target`

Introduced before R2006a

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 | `SimulinkRealTime.target`

Introduced before R2006a

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 | `SimulinkRealTime.target`

Introduced before R2006a

xPCNumLogSamples

Return number of samples in log buffer

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.

Return

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 logged.

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

See Also

xPCGetStateLog | xPCGetOutputLog | xPCGetTETLog | xPCGetTimeLog | xPCMaxLogSamples

Introduced before R2006a

xPCNumLogWraps

Return number of times log buffer wraps

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.

Return

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 | `SimulinkRealTime.target`

Introduced before R2006a

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 represented 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

Introduced before R2006a

xPCOpenTcpIpPort

Open TCP/IP connection to Simulink Real-Time system

Prototype

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

Arguments

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

Return

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. If xPCOpenTcpIpPort succeeds, it returns a nonnegative integer. 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

Introduced before R2006a

xPCReboot

Restart 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. `xPCReboot` returns nothing. This function does not close the connection to the target computer. After the target computer restarts, either explicitly close the port or call xPCReOpenPort.

See Also

xPCReOpenPort | `SimulinkRealTime.target`

Introduced before R2006a

xPCReOpenPort

Reopen communication channel

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.

Return

If the function reopens a connection without detecting an error, it returns `0`. If it 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 existing settings, while the other functions first set up the port.

See Also

xPCOpenTcpIpPort | xPCClosePort

Introduced before R2006a

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_TCPIP`.
	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

Introduced before R2006a

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 | SimulinkRealTime.target

Introduced before R2006a

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 that contains 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.

If you change parameter values while the application is running in `Real-Time` mode, save your real-time application parameter values. By using the saved values, you can recreate real-time application parameter settings from various runs.

See Also

xPCLoadParamSet

Introduced before R2006a

xPCScAddSignal

Add signal to scope

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
scNum	Enter the scope number.
sigNum	Enter a signal number.

Description

The `xPCScAddSignal` function adds the signal with number *`sigNum`* to scope *`scNum`*. The signal cannot 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`

Introduced before R2006a

xPCScGetAutoRestart

Scope autorestart status

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
scNum	Enter the scope number.

Return

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

Introduced in R2009b

xPCScGetData

Copy scope data to array

Prototype

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

Arguments

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

Description

The `xPCScGetData` function gets the data used in a scope. Use this function for scopes of type `SCTYPE_HOST`. The scope must be either in state `"Finished"` or in state `"Interrupted"` for the data to be retrievable. (Use the xPCScGetState function to check the state of the scope.) The data must be retrieved one signal at a time. The calling function must allocate the space ahead of time to store the scope data. *data* must be an array of doubles, regardless of the data type of the signal to be retrieved. Use the function xPCScGetSignals to get the list of signals in the scope for *signal_id*. Use the function xPCGetScope to get the scope number for *scNum*.

To get timestamped 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`

Introduced before R2006a

xPCScGetDecimation

Return decimation of scope

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
scNum	Enter the scope number.

Return

Returns the decimation of scope *scNum*. If the function detects an error, it returns -1.

Description

The xPCScGetDecimation function gets the decimation of scope *scNum*. The decimation is a number, N, meaning every Nth sample is acquired in a scope window. Use the xPCGetScope function to get the scope number.

See Also

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

Introduced before R2006a

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

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`

Introduced before R2006a

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

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`

Introduced before R2006a

xPCScGetNumSignals

Get number of signals in scope

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
scNum	Enter the scope number.

Return

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`

Introduced in R2008b

xPCScGetSignalList

Copy list of signals to array

Prototype

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

Arguments

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

Description

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

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

Introduced in R2008b

xPCScGetSignals

Copy list of signals to array

Prototype

```
void xPCScGetSignals(int port, int scNum, int *data);
```

Arguments

port	Value returned by the function xPCOpenTcpIpPort.
scNum	Enter the scope number.
data	Integer array allocated by the caller as a list containing the signal identifiers, terminated by -1.

Description

The xPCScGetSignals function gets the list of signals defined for scope *scNum*. You can use the constant MAX_SIGNALS, defined in xpcapiconst.h, as the size of *data*. Use the xPCGetScope function to get the scope number.

Note This function will be removed in a future release. Use the xPCScGetSignalList function instead.

See Also

xPCScGetData | xPCGetScopes | Real-Time File Scope | Real-Time Host Scope | Real-Time Target Scope

Introduced before R2006a

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

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. `xPCScGetStartTime` 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`

Introduced before R2006a

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

Returns the state of scope *scNum*. If the function detects an error, it returns `-1`.

Description

The `xPCScGetState` function gets the state of scope *scNum*, or `-1` upon error. Use the xPCGetScope function to get the scope number.

Constants to find the scope state, defined in `xpcapiconst.h`, have the following meanings:

Constant	Value	Description
SCST_WAITTOSTART	0	Scope is ready and waiting to start.
SCST_PREACQUIRING	5	Scope acquires a predefined number of samples before triggering.

Constant	Value	Description
SCST_WAITFORTRIG	1	After a scope is finished with the preacquiring state, it waits for a trigger. If the scope does not preacquire data, it enters the wait for trigger state.
SCST_ACQUIRING	2	Scope is acquiring data. The scope enters this state when it leaves the wait for trigger state.
SCST_FINISHED	3	Scope is finished acquiring data when it has attained the predefined limit.
SCST_INTERRUPTED	4	You stopped (interrupted) the scope.

See Also

xPCScStop | xPCScStart | `Real-Time File Scope` | `Real-Time Host Scope` | `Real-Time Target Scope`

Introduced before R2006a

xPCScGetTriggerLevel

Get trigger level for scope

Prototype

```
double xPCScGetTriggerLevel(int port, int scNum);
```

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
scNum	Enter the scope number.

Return

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`

Introduced before R2006a

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

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	Another scope triggers this scope at the trigger point of the triggering scope, modified by the value of `triggerscopesample` (see scopedata).

See Also

xPCScSetTriggerMode | xPCScGetTriggerScope | xPCScSetTriggerScope | xPCScGetTriggerSignal | xPCScSetTriggerSignal | xPCScGetTriggerSlope | xPCScSetTriggerSlope | xPCScGetTriggerLevel | xPCScSetTriggerLevel | xPCGetScope | `Real-Time File Scope` | `Real-Time Host Scope` | `Real-Time Target Scope`

Introduced before R2006a

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

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`

Introduced before R2006a

xPCScGetTriggerScopeSample

Get sample number for triggering scope

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
scNum	Enter the scope number.

Return

If the function acquires a real sample, it returns a nonnegative integer. If the triggering scope triggers at the end of the data acquisition cycle, the function returns `-1`. 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. 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`

Introduced before R2006a

xPCScGetTriggerSignal

Get trigger signal for scope

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
scNum	Enter the scope number.

Return

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`

Introduced before R2006a

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

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`

Introduced before R2006a

xPCScGetType

Get type of scope

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
scNum	Enter the scope number.

Return

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`

Introduced before R2006a

xPCScRemSignal

Remove signal from scope

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
scNum	Enter the scope number.
sigNum	Enter a signal number.

Description

The `xPCScRemSignal` function removes a signal from the scope with number *scNum*. The scope must 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`

Introduced before R2006a

xPCScSetAutoRestart

Scope autorestart status

Prototype

```
void xPCScSetAutoRestart(int port, int scNum, int autorestart)
```

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
scNum	Enter the scope number.
autorestart	To enable scope autorestart, enter `1`. To disable scope autorestart, enter `0`).

Description

The `xPCScSetAutoRestart` function sets the autorestart flag for scope *scNum* to `0` or `1`. The value `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`

Introduced in R2009b

xPCScSetDecimation

Set decimation of scope

Prototype

```
void xPCScSetDecimation(int port, int scNum, int decimation);
```

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
scNum	Enter the scope number.
decimation	Enter an integer for the decimation.

Description

The `xPCScSetDecimation` function sets the decimation of scope *scNum*. The decimation is a number, `N`, meaning every `N`th sample is acquired in a scope window. Use this function only when the scope is stopped. Use xPCScGetState to check the state of the scope. Use the xPCGetScope function to get the scope number.

See Also

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

Introduced before R2006a

xPCScSetNumPrePostSamples

Set number of pre- or posttriggering samples before triggering scope

Prototype

```
void xPCScSetNumPrePostSamples(int port, int scNum, int prepost);
```

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
scNum	Enter the scope number.
prepost	A negative number means pretriggering, while a positive number means posttriggering. This function can only be used when the scope is stopped.

Description

The `xPCScSetNumPrePostSamples` function sets the number of samples for pre- or posttriggering for scope *scNum* to *prepost*. Use this function only when the scope is stopped. Use xPCScGetState to check the state of the scope. Use the xPCGetScope function to get the scope number.

See Also

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

Introduced before R2006a

xPCScSetNumSamples

Set number of samples in one data acquisition cycle

Prototype

```
void xPCScSetNumSamples(int port, int scNum, int samples);
```

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
scNum	Enter the scope number.
samples	Enter the number of samples you want to acquire in one cycle.

Description

The `xPCScSetNumSamples` function sets the number of samples for scope *scNum* to *samples*. Use this function only when the scope is stopped. Use xPCScGetState to check the state of the scope. Use the xPCGetScope function to get the scope number.

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

- Autorestart is on — When the scope triggers, the scope starts collecting 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 starts collecting data into a memory buffer. It stops when it has collected the number of samples that you specified. 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`

Introduced before R2006a

xPCScSetTriggerLevel

Set trigger level for scope

Prototype

```
void xPCScSetTriggerLevel(int port, int scNum, double level);
```

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
scNum	Enter the scope number.
level	Value for a signal to trigger data acquisition with a scope.

Description

The `xPCScSetTriggerLevel` function sets the trigger level to *`level`* for scope *`scNum`*. Use this function only when the scope is stopped. Use xPCScGetState to check the state of the scope. Use the xPCGetScope function to get the scope number for the trigger scope.

See Also

xPCScGetTriggerSlope | xPCScSetTriggerSignal | xPCScGetTriggerSignal | xPCScSetTriggerScope | xPCScGetTriggerScope | xPCScSetTriggerMode | xPCScGetTriggerMode | xPCScGetState | xPCScSetTriggerSlope | xPCScGetTriggerLevel | xPCGetScope | `Real-Time Host Scope` | `Real-Time File Scope` | `Real-Time Target Scope`

Introduced before R2006a

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 mode 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.

Constant	Value	Description
TRIGMD_SCOPE	3	Another scope triggers this scope at the trigger point of the triggering scope, modified by the value of `triggerscopesample` (see scopedata).

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`

Introduced before R2006a

xPCScSetTriggerScope

Select scope for triggering another scope

Prototype

```
void xPCScSetTriggerScope(int port, int scNum, int trigScope);
```

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
scNum	Enter the scope number.
trigScope	Enter the scope number of the scope used for a trigger.

Description

The `xPCScSetTriggerScope` function sets the trigger scope of scope *scNum* to *trigScope*. This function can only be used when the scope is stopped. Use xPCScGetState to check the state of the scope. Use the xPCGetScopes function to get a list of scopes.

The scope type can be `SCTYPE_HOST`, `SCTYPE_TARGET`, or `SCTYPE_FILE`.

See Also

xPCGetScopes | xPCScSetTriggerLevel | xPCScGetTriggerLevel | xPCScSetTriggerSlope | xPCScGetTriggerSlope | xPCScSetTriggerSignal | xPCScGetTriggerSignal | xPCScGetTriggerScope | xPCScSetTriggerMode | xPCScGetTriggerMode | xPCScGetState | xPCGetScope | `Real-Time Host Scope` | `Real-Time File Scope` | `Real-Time Target Scope`

Introduced before R2006a

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`

Introduced before R2006a

xPCScSetTriggerSignal

Select signal for triggering scope

Prototype

```
void xPCScSetTriggerSignal(int port, int scNum, int trigSig);
```

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
scNum	Enter the scope number.
trigSig	Enter a signal number.

Description

The `xPCScSetTriggerSignal` function sets the trigger signal of scope *`scNum`* to *`trigSig`*. The trigger signal *`trigSig`* must be one of the signals in the scope. Use this function only when the scope is stopped. You can use xPCScGetSignals to get the list of signals in the scope. Use xPCScGetState to check the state of the scope. Use the xPCGetScopes function to get a list of scopes.

See Also

xPCGetScopes | xPCScGetState | xPCScSetTriggerLevel | xPCScGetTriggerLevel | xPCScSetTriggerSlope | xPCScGetTriggerSlope | xPCScGetTriggerSignal | xPCScSetTriggerScope | xPCScGetTriggerScope | xPCScSetTriggerMode | xPCScGetTriggerMode | xPCGetScope | `Real-Time Host Scope` | `Real-Time File Scope` | `Real-Time Target Scope`

Introduced before R2006a

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`

Introduced before R2006a

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`

Introduced before R2006a

xPCScStart

Start data acquisition for scope

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
scNum	Enter the scope number.

Description

The `xPCScStart` function starts or restarts the data acquisition of scope *`scNum`*. If the scope does not have to preacquire samples, it enters the `Waiting for Trigger` state. The scope must be in state `Waiting to Start`, `Finished`, or `Interrupted` for this function to succeed. To check the state of the scope, call xPCScGetState. 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`

Introduced before R2006a

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`* and 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`

Introduced before R2006a

xPCSetEcho

Turn message display on or off

Prototype

```
void xPCSetEcho(int port, int mode);
```

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.	
mode	Valid values are	
	`0`	Turns off the display
	`1`	Turns on the display

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 off the message display, the message screen no longer updates. Existing messages remain on the screen as they were.

See Also

xPCGetEcho

Introduced before R2006a

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 function is useful only to set the character string constant to ENOERR, indicating no error was found.

See Also

xPCGetLastError | xPCErrorMsg

Introduced before R2006a

xPCSetLoadTimeOut

Change initialization timeout value between development and target computers

Prototype

```
void xPCSetLoadTimeOut(int port, int timeOut);
```

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
timeOut	Enter the new communication timeout value.

Description

The `xPCSetLoadTimeOut` function changes the timeout value for communication between the development and target computers. The *`timeOut`* value is the time a Simulink Real-Time API function waits for the communication to complete before returning. It enables you to set the number of communication attempts to be made before signaling a timeout.

See Also

xPCLoadApp | xPCGetLoadTimeOut | xPCUnloadApp | `SimulinkRealTime.target`

Introduced before R2006a

xPCSetLogMode

Set logging mode and increment value of scope

Prototype

```
void xPCSetLogMode(int port, lgmode logging_data);
```

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
logging_data	Logging mode and increment value.

Description

The xPCSetLogMode function sets the logging mode and increment to the values set in *logging_data*. See the structure lgmode for more details.

See Also

lgmode | xPCGetLogMode | SimulinkRealTime.target

Introduced before R2006a

xPCSetParam

Change value of parameter

Prototype

```
void xPCSetParam(int port, int paramIdx, const double *paramValue);
```

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
paramIdx	Parameter index.
paramValue	Vector of doubles, assumed to be the size required by the parameter type

Description

The xPCSetParam function sets the parameter *paramIdx* to the value in *paramValue*. *paramValue* can contain a matrix represented as a vector 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

Introduced before R2006a

xPCSetSampleTime

Change real-time application sample time

Prototype

```
void xPCSetSampleTime(int port, double ts);
```

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
ts	Sample time for the real-time application.

Description

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

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 | `SimulinkRealTime.target`

Introduced before R2006a

xPCSetScope

Set properties of scope

Prototype

```
void xPCSetScope(int port, scopedata state);
```

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
state	Enter a structure of type `scopedata`.

Description

Note The `xPCSetScope` function will be removed in a future release. Use the `xPCScSet`*`ScopePropertyName`* functions to access property values instead. For example, to set the number of samples to acquire in one data acquisition cycle, use `xPCScSetNumSamples`.

The `xPCSetScope` function sets the properties of a scope using a *`state`* structure of type scopedata. Set the properties you want to set for the scope. You can set several properties at the same time. For convenience, call the function xPCGetScope first to populate the structure with the current values. You can then change the desired values. Use this function only when the scope is stopped. Use xPCScGetState to determine the state of the scope.

See Also

xPCGetScope | xPCScGetState | scopedata

Introduced before R2006a

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 runs for this number of seconds before stopping. Set *`tfinal`* to `-1.0` to set the stop time to infinity.

See Also

xPCGetStopTime | `SimulinkRealTime.target`

Introduced before R2006a

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 | `SimulinkRealTime.target`

Introduced before R2006a

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 | `SimulinkRealTime.target`

Introduced before R2006a

xPCTargetPing

Ping target computer

Prototype

```
int xPCTargetPing(int port);
```

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.

Return

If the target responds, the function returns `1`. If the target computer does not respond, the function returns `0`. The function does not return an error status.

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. Such errors include an invalid port number or the port is not open. Other errors, such as the inability to connect to the target, are ignored.

`xPCTargetPing` causes 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 a failed connection. For example, if the program running on your development computer has a fatal error and aborts its I/O connection, you can close it.

See Also

xPCOpenConnection | xPCOpenTcpIpPort | xPCClosePort

Introduced in R2007a

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 that the grid is on, while `0` implies that the grid is off. 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`

Introduced before R2006a

xPCTgScGetMode

Get scope mode for displaying signals

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
scNum	Enter the scope number.

Return

Returns the value corresponding to the scope mode. The possible values are

- `SCMODE_NUMERICAL = 0`
- `SCMODE_REDRAW = 1`
- `SCMODE_SLIDING = 2` will be removed in a future release. It behaves like value `SCMODE_ROLLING = 3`.
- `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_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`

Introduced before R2006a

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`

Introduced before R2006a

xPCTgScGetYLimits

Copy *y*-axis limits for scope to array

Prototype

```
void xPCTgScGetYLimits(int port, int scNum, double *limits);
```

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
scNum	Enter the scope number.
limits	The first element of the array is the lower limit while the second element is the upper limit.

Description

The xPCTgScGetYLimits function gets and copies the upper and lower limits for a scope of type SCTYPE_TARGET and with scope number *scNum*. The limits are stored in the array *limits*. If both elements are zero, the limits are autoscaled. Use the xPCGetScopes function to get a list of scopes.

See Also

xPCGetScopes | xPCTgScSetGrid | xPCTgScGetGrid | xPCTgScSetViewMode | xPCTgScGetViewMode | xPCTgScSetMode | xPCTgScGetMode | xPCTgScSetYLimits | Real-Time Target Scope

Introduced before R2006a

xPCTgScSetGrid

Set grid mode for scope

Prototype

```
void xPCTgScSetGrid(int port, int scNum, int grid);
```

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
scNum	Enter the scope number.
grid	Enter a grid value.

Description

The xPCTgScSetGrid function sets the grid of a scope of type SCTYPE_TARGET and scope number *scNum* to *grid*. If *grid* is 0, the grid is off. If *grid* is 1, the grid is on and grid lines are drawn on the scope window. When the drawing mode of scope *scNum* is set to SCMODE_NUMERICAL, the grid is not drawn even when the grid mode is set to 1. Use the xPCGetScopes function to get a list of scopes.

See Also

xPCGetScopes | xPCTgScGetGrid | xPCTgScSetViewMode | xPCTgScGetViewMode | xPCTgScSetMode | xPCTgScGetMode | xPCTgScSetYLimits | xPCTgScGetYLimits | Real-Time Target Scope

Introduced before R2006a

xPCTgScSetMode

Set display mode for scope

Prototype

```
void xPCTgScSetMode(int port, int scNum, int mode);
```

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
scNum	Enter the scope number.
mode	Enter the value for the mode.

Description

The `xPCTgScSetMode` function sets the mode of a scope of type `SCTYPE_TARGET` and scope number *`scNum`* to *`mode`*. You can use one of the following constants for *`mode`*:

- `SCMODE_NUMERICAL = 0`
- `SCMODE_REDRAW = 1`
- `SCMODE_SLIDING = 2` will be removed in a future release. It behaves like value `SCMODE_ROLLING = 3`.
- `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`

Introduced before R2006a

xPCTgScSetViewMode

Set view mode for scope

Prototype

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

Arguments

port	Enter the value returned by the function xPCOpenTcpIpPort.
scNum	Enter the scope number.

Description

Note `xPCTgScSetViewMode` has no function.

See Also

xPCGetScopes | xPCTgScSetGrid | xPCTgScGetGrid | xPCTgScGetViewMode | xPCTgScSetMode | xPCTgScGetMode | xPCTgScSetYLimits | xPCTgScGetYLimits | `Real-Time Target Scope`

Introduced before R2006a

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`

Introduced before R2006a

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 target computer memory, and prepares the target computer for receiving a new real-time application. The function xPCLoadApp calls this function before loading a new real-time application.

See Also

xPCLoadApp | `SimulinkRealTime.target`

Introduced before R2006a

6

MATLAB API

macaddr

Convert character vector or string scalar MAC address to vector-based address

Syntax

```
macaddr(MAC_address)
```

Description

`macaddr(MAC_address)` converts a character vector or string scalar 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

character vector | string scalar

The value is entered as a character vector or string scalar comprised of six colon-delimited fields of two-digit hexadecimal numbers.

Example: '01:23:45:67:89:ab'

Data Types: `char` | `string`

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.

Before calling `profile_slrt`, use the profiler API functions to start and stop the profiler.

In a future release, the `profile_slrt` function will be removed. Use the profiler API functions to start and stop the profiler and to display the profiler data.

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 lists the model sections. The numbers underneath the bars indicate the processor cores.

The Code Execution Profiling Report displays model execution profile results for each task.

- To display the profile data for a section of the model, click the **Membrane** button next to the section.
- To display the TET data for the section in the Simulation Data Inspector, click the **Plot time series data** button .
- To view the section in Simulink Editor, click the link next to the **Expand Tree** button [+].
- To view the lines of generated code corresponding to the section, click the **Expand Tree** button [+] and then click the **View Source** button .

In the **Verification** tab of the Code Generation dialog box, the **Measure task execution time** check box is checked and locked. 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

Function Profiling Example

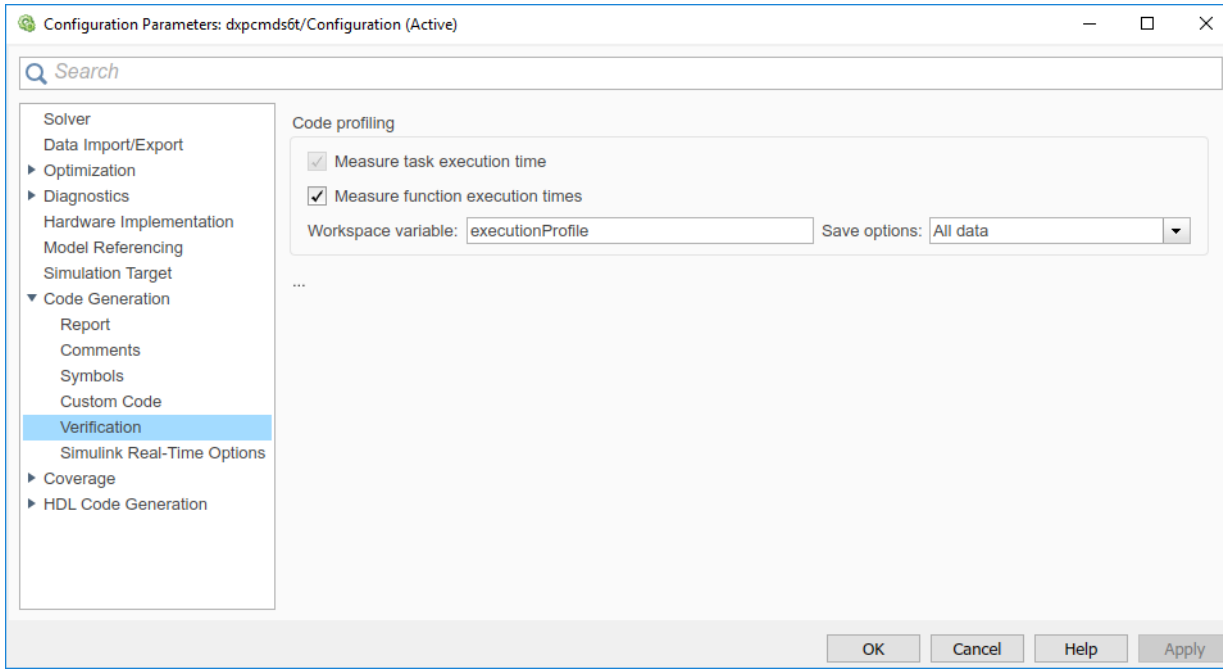
Profile the concurrent execution model `dxpcmds6t` on a multicore target computer with function profiling.

Open model `dxpcmds6t`.

```
open_system('dxpcmds6t');
```

In the top model, open the **Configuration Parameters**. In the **Real-Time** tab, click **Hardware Settings**. Select **Code Generation > Verification**.

Select the **Measure function execution times** check box.



Build and download the model.

```
rtwbuild('dxpcmds6t');
tg = slrt('TargetPC1');
load(tg,'dxpcmds6t');
```

Start the profiler and execute the model for 2 s.

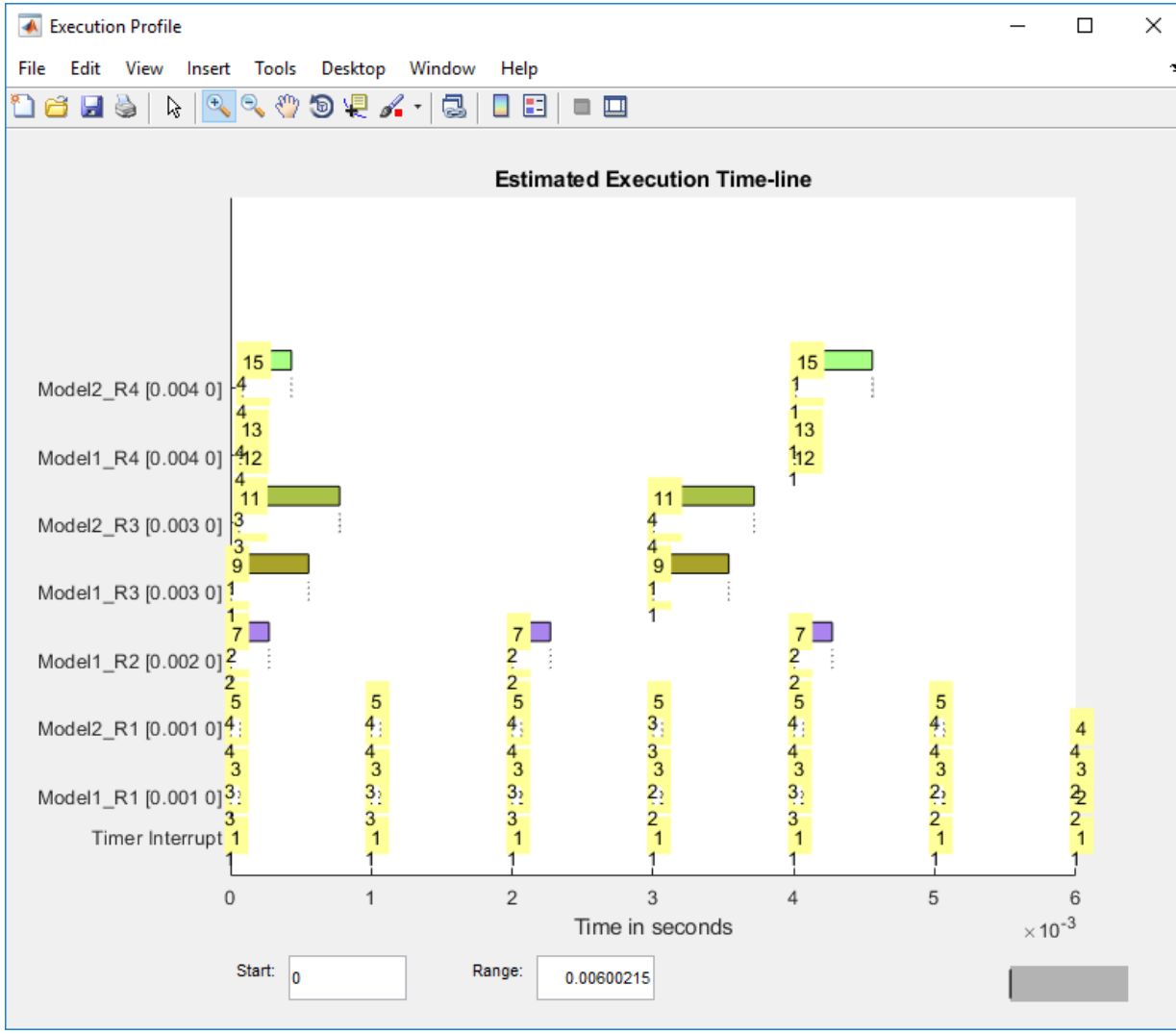
```
startProfiler(tg);
start(tg);
pause(2);
stop(tg);
```

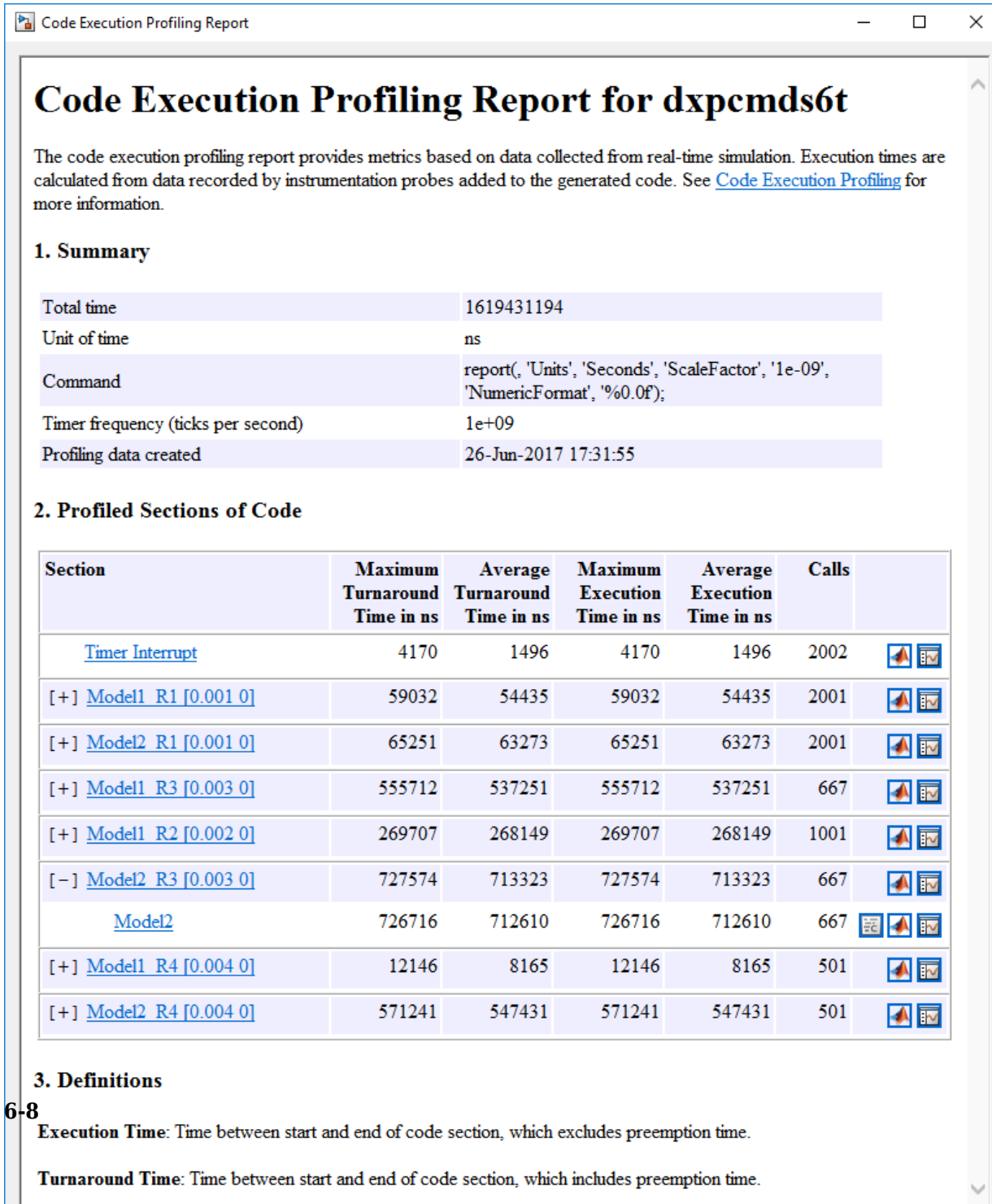
The stop execution command also stops the profiler.

Profile the real-time application execution.

```
profileInfo.modelname = 'dxpcmds6t.slx';
profData = profile_slrt(profileInfo);
```

For more information about the time line in the execution profile plot, see `timeline`.


Execution Profile
File Edit View Insert Tools Desktop Window Help
Estimated Execution Time-line
Model2_R4 [0.004 0]
Model1_R4 [0.004 0]
Model2_R3 [0.003 0]
Model1_R3 [0.003 0]
Model1_R2 [0.002 0]
Model2_R1 [0.001 0]
Model1_R1 [0.001 0]
Timer Interrupt
0 1 2 3 4 5 6
Time in seconds
×10-3
Start: 0
Range: 0.00600215



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	1619431194
Unit of time	ns
Command	report(, 'Units', 'Seconds', 'ScaleFactor', '1e-09', 'NumericFormat', '%0.0f');
Timer frequency (ticks per second)	1e+09
Profiling data created	26-Jun-2017 17:31:55

2. Profiled Sections of Code

Section	Maximum Turnaround Time in ns	Average Turnaround Time in ns	Maximum Execution Time in ns	Average Execution Time in ns	Calls
Timer Interrupt	4170	1496	4170	1496	2002
[+] Model1_R1 [0.001 0]	59032	54435	59032	54435	2001
[+] Model2_R1 [0.001 0]	65251	63273	65251	63273	2001
[+] Model1_R3 [0.003 0]	555712	537251	555712	537251	667
[+] Model1_R2 [0.002 0]	269707	268149	269707	268149	1001
[−] Model2_R3 [0.003 0]	727574	713323	727574	713323	667
Model2	726716	712610	726716	712610	667
[+] Model1_R4 [0.004 0]	12146	8165	12146	8165	501
[+] Model2_R4 [0.004 0]	571241	547431	571241	547431	501

3. Definitions

Execution Time: Time between start and end of code section, which excludes preemption time.

Turnaround Time: Time between start and end of code section, which includes preemption time.

Profile configuration data, consisting of the following fields:

`rawdataonhost` — Flag specifying whether the raw data is on development or target computer
`0` (default) | `1`

- `0` — The raw data file `xPCTrace.csv` is on the target computer. Transfer the file from the target computer to the host.
- `1` — The raw data file `xPCTrace.csv` is in the current folder on the development computer.

Data Types: `double`

`modelname` — Name of the model to be profiled
usrname

The name can include the model file extension.

Data Types: `char`

`noplot` — Flag suppressing execution profile plot
`0` (default) | `1`

- `0` — Display the execution profile plot on the development computer monitor.
- `1` — Do not display the execution profile plot on the development computer monitor.

Data Types: `double`

`noreport` — Flag suppressing code execution profiling report
`0` (default) | `1`

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

Data Types: `double`

Output Arguments

`profData` — Profile results data
structure

Profile results data stored in an object of type `coder.profile.ExecutionTime`.

`TimerTicksPerSecond` — Number of seconds per timer tick
double

Scales the execution-time tick.

`Sections` — Array of results data for profiled code sections
array

Each array item is an object of type `coder.profile.ExecutionTimeSection`.

See Also

`Profiler Data` | `Sections` | `TimerTicksPerSecond` | `getProfilerData` | `resetProfiler` | `startProfiler` | `stopProfiler` | `timeline`

Topics

“Execution Profiling for Real-Time Applications”
“Troubleshoot Failed Read of Profiling Data”

Introduced in R2014a

slrt

Interface for managing target computer

Syntax

```
target_object = slrt
target_object = slrt(target_name)
```

Description

`target_object = slrt` constructs a target object representing the default target computer. Select the default target computer by using the `SimulinkRealTime.utils.TargetComputerManager`.

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, check 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. Select the default target computer by using the `SimulinkRealTime.utils.TargetComputerManager`.

```
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 | string scalar

Example: `'TargetPC1'`

Data Types: `char` | `string`

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`

See Also

`SimulinkRealTime.target` | Target Settings

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.

When you run Simulink Real-Time Explorer from within MATLAB, you have available the full capabilities of 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 it as a standalone executable, you have available a subset of the capabilities of Simulink Real-Time Explorer.

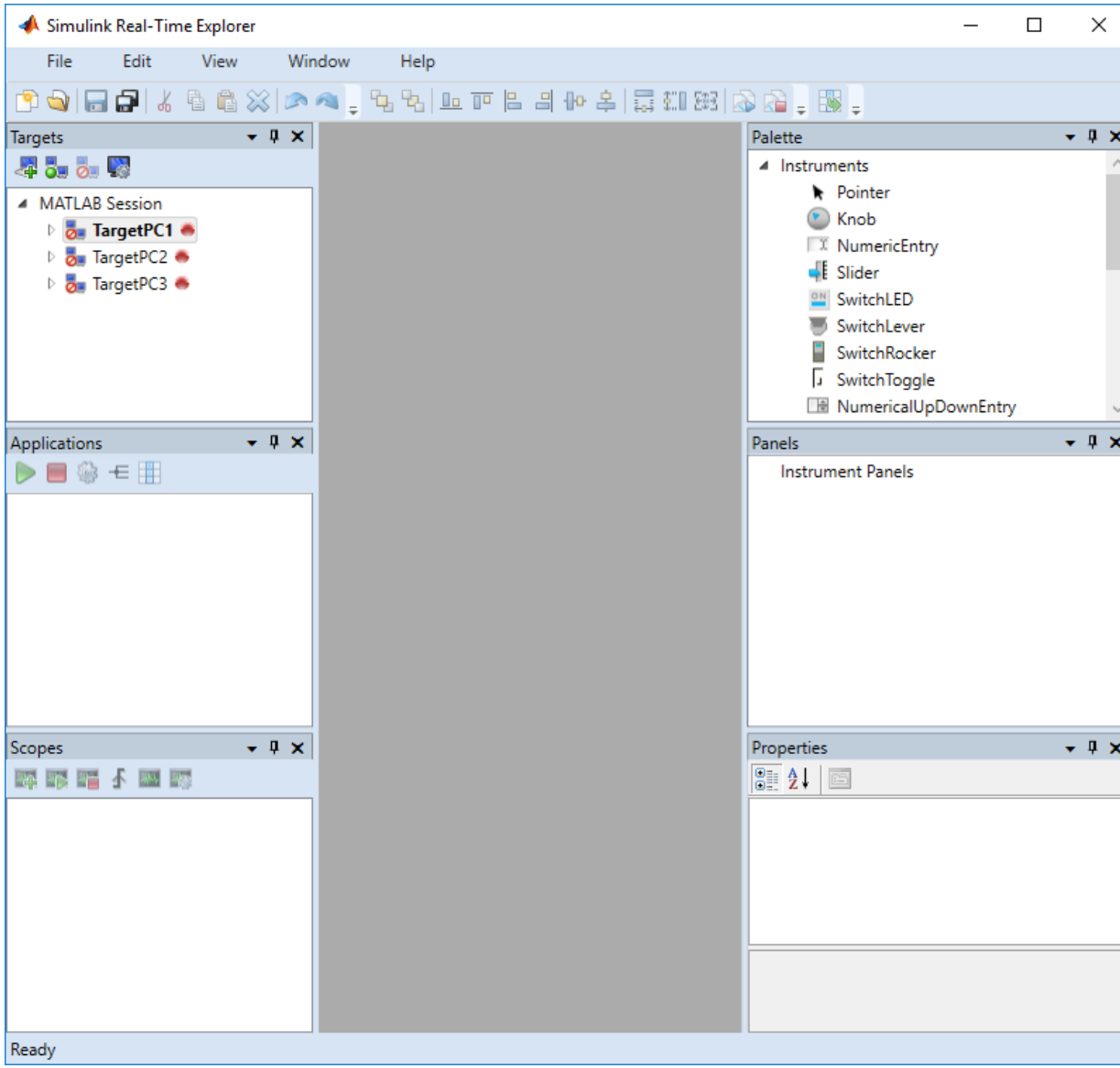
For more information, see **Simulink Real-Time Explorer**.

Examples

Default

Open Simulink Real-Time Explorer

```
slrtexplr
```



See Also

Simulink Real-Time Explorer

Introduced in R2014a

SimulinkRealTime.prototype.Explorer

Interact with target computer and real-time application running on target computer

Syntax

```
SimulinkRealTime.prototype.Explorer
```

Description

`SimulinkRealTime.prototype.Explorer` opens the Target Computer Explorer.

Simulink Real-Time Target Computer Explorer provides a single point of contact for viewing connection status and interacting with a real-time application. You can monitor and trace signals, tune parameters, and stream data to the Simulation Data Inspector.

Examples

Select Signals and Stream Data to Simulation Data Inspector

This example shows how to connect to the target computer, load the real-time application, select signals for a signal list, start the real-time application, and view the streaming data in the Simulation Data Inspector..

Open the Simulink Real-Time Target Computer Explorer, type:

```
SimulinkRealTime.prototype.Explorer
```

In the Simulink Real-Time Target Computer Explorer, to connect to the selected target computer, click **Connect**.

To select and load a real-time application, click **Load Application** and select the `mldatx` file.

To select signals for streaming, click the application name, select signals from the **Signals** tab, and click the **Add selected signals** button.

To run the application and generate data for streaming, click the **Run** button.

To stream the signal data, select the signals in the **Group signals to stream for SDI** list and click the **Stream Signal Group to SDI** button.

To view the streaming signals, click the **Open in SDI** button.

After viewing the data, to stop the real-time application, click the **Stop** button.

See Also

`SimulinkRealTime.utils.TargetComputerManager` | `slrtexplr`

Topics

Simulink Real-Time Explorer
SimulinkRealTime.prototype.Explorer
Simulink Real-Time Target Computer Manager

Introduced in R2019a

SimulinkRealTime.utils.TargetComputerManager

Configure connection to target computer and real-time application for execution

Syntax

```
SimulinkRealTime.utils.TargetComputerManager.open
```

Description

`SimulinkRealTime.utils.TargetComputerManager.open` opens the Target Computer Manger.

When you open the Simulink Real-Time Target Computer Manager, you can configure the interface to Target Computers.

For more information, see **Simulink Real-Time Target Computer Manager**.

Examples

Open the Target Computer Manager

Open the Simulink Real-Time Target Computer Manager.

- In the Simulink Editor, on the **Real-Time** tab, from target computers list in the **Connect to Target Computer** section, click **Target Computer Manager**.
- In the MATLAB Command Window, type:

```
SimulinkRealTime.utils.TargetComputerManager.open
```

See Also

Simulink Real-Time Explorer | **Simulink Real-Time Target Computer Manager** | `slrtexplr`

Introduced in R2019a

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.

In a future release, the `slrtsetCC` and `slrtgetCC` functions will be removed.

Use the `mex` command. The `slrtgetCC` and `slrtsetCC` functions do not support the Visual Studio 2019 compiler. To set up this compiler for Simulink Real-Time, use `mex -setup`. For more information, see `slrtsetCC`.

`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
    ocation = C:\Program Files (x86)\Microsoft Visual Studio 14.0\
```

Return Compiler Type

```
type = slrtgetCC('Type')
```

```
type =

VisualC
```

Return Compiler Location

```
location = slrtgetCC('Location')
```

```
location =

'C:\Program Files (x86)\Microsoft Visual Studio 14.0\'
```

Return Compiler Type and Location

```
[type, location] = slrtgetCC

type =

    'VisualC'


location =

    'C:\Program Files (x86)\Microsoft Visual Studio 14.0\'
```

Display Supported Compilers

```
slrtgetCC('supported')

List of C++ Compilers supported by Simulink Real-Time:

Name                                              Version        Service Packs
Microsoft Visual Studio 2015                      14.0
Microsoft Visual Studio 2017                      15.0
2015 toolset in Microsoft Visual Studio 2017      14.0
```

Display Supported Compilers Installed

```
slrtgetCC('installed')

List of installed C++ Compilers:

Name: Microsoft Visual Studio 2015
Location: C:\Program Files (x86)\Microsoft Visual Studio 14.0\
```

Return Supported Compilers Installed

```
[compilers] = slrtgetCC('installed')
compilers(1)
```

```
compilers =

  struct with fields:

       Type: 'VisualC'
       Name: 'Microsoft Visual Studio 2015'
    Location: 'C:\Program Files (x86)\Microsoft Visual Studio 14.0\'


ans =

  struct with fields:

       Type: 'VisualC'
       Name: 'Microsoft Visual Studio 2015'
    Location: 'C:\Program Files (x86)\Microsoft Visual Studio 14.0\'
```

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

See Also

`mex` | `slrtsetCC`

External Websites

www.mathworks.com/support/compilers/current_release

Introduced in R2014a

slrtpingtarget

Test communication between development and target computers

Syntax

```
link_status = slrtpingtarget
link_status = slrtpingtarget(target_object)
link_status = slrtpingtarget(target_computer_name)
[link_status connection_info] = slrtpingtarget( ___ )

[link_status connection_info] = slrtpingtarget( ___ , 'info')

[link_status connection_info] = slrtpingtarget( ___ , 'reset')
```

Description

`link_status = slrtpingtarget` without an argument tests at a low level whether the development computer and the default target computer can communicate using the settings for that target computer. If a data channel is open between the development and target computers, the function leaves it open.

`link_status = slrtpingtarget(target_object)` tests whether the development computer and the target computer represented by `target_object` can communicate using the settings stored in `target_object`. If a data channel is open between the development and target computers, the function leaves it open.

`link_status = slrtpingtarget(target_computer_name)` tests whether the development computer can communicate with target computer `target_computer_name` using the settings for that target computer. If a data channel is open between the development and target computers, the function leaves it open.

Calls to `[link_status connection_info] = slrtpingtarget( ___ )` have the same behavior as `slrtpingtarget(target_object)`.

`[link_status connection_info] = slrtpingtarget( ___ , 'info')` uses the information/control channel to return information about the Simulink Real-Time

connection between the development and target computers. If a data channel is open between the development and target computers, the function leaves it open.

`[link_status connection_info] = slrtpingtarget( ___ , 'reset')` uses the information/control channel to close an open communication channel between the development and target computers and then returns link status and connection information.

Examples

Check Communication with Default Target Computer

```
link_status = slrtpingtarget
```

```
link_status =

success
```

Check Communication with Target Computer by Target Object

```
target_object = slrt('TargetPC1');
link_status = slrtpingtarget(target_object)
```

```
link_status =

success
```

Check Communication with Target Computer by Name

```
link_status = slrtpingtarget('TargetPC1')
```

```
link_status =

failed
```

Get Information About Default Target Computer Connection

```
[link_status connection_info] = slrtpingtarget('info')

link_status =

success

connection_info =

10.10.10.100
```

Get Information About Connection with Target Object

```
target_object = slrt('TargetPC1');
[link_status connection_info] = ...
     slrtpingtarget(target_object, 'info')

link_status =

success

connection_info =

Disconnected
```

Get Information About Target Computer with Name

```
[link_status connection_info] = slrtpingtarget('TargetPC1', 'info')

link_status =

failed

connection_info =

'fail: Target machine did not respond.'
```

Reset Default Target Computer

```
[link_status connection_info] = slrtpingtarget('reset')
```

```
link_status =

success

connection_info =

Disconnected
```

Input Arguments

`target_computer_name` — Name of specific target computer
`'TargetPC1'` (default) | character vector | string scalar | ...

Name property of a particular target computer environment object. The default name is `'TargetPC1'`.

Example: `'TargetPC2'`

Data Types: `char` | `string`

`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`

Output Arguments

`link_status` — Reports if communication is possible between the development and target computers
`'success'` | `'failed'`

- If communication is possible between the development and target computers, this value is `'success'`. The value `'success'` does not mean that Simulink Real-Time has established a connection, only that one is possible.

- If communication is not possible between the development and target computers, this value is `'failed'`. The function returns `'failed'` for such reasons as a faulty or disconnected Ethernet cable or an erroneous IP address setting. For more information, see “Troubleshoot Communication Failure with Target Computers”.

`connection_info` — Reports whether a connection is active to a development computer network address
`'xx:xx:xx:xx'` | `'Disconnected'` | character vector

If you call `ping` without a second argument:

- If communication is possible, `connection_info` is empty.
- If communication is not possible, `connection_info` contains an error message.

If you call `ping` with a second argument of `'info'`:

- If the connection is active, `connection_info` reports the development computer network address to which the target computer is connected.
- If the connection is not active, `connection_info` contains `'Disconnected'`.
- If communication is not possible, `connection_info` contains an error message.

If you call `ping` with a second argument of `'reset'`:

- If communication is possible, `connection_info` contains `'Disconnected'`.
- If communication is not possible, `connection_info` contains an error message.

See Also
`SimulinkRealTime.target` | `ping`

Topics
“Troubleshoot Communication Failure with Target Computers”

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.

In a future release, the `slrtsetCC` and `slrtgetCC` functions will be removed.

Use the `mex` command. 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. The `slrtsetCC` function does not support the Visual Studio 2019 compiler. To set up this compiler, see “Select Visual Studio 2019 Compiler” on page 6-32.

By default, the Microsoft Visual Studio 2015 installer does not install the C++ compiler that Simulink Real-Time requires. To install the C++ compiler, perform a custom install and select the C++ compiler. If you have already installed Microsoft Visual Studio with the default configuration, rerun the installer and select the modify option.

`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

Select Compiler

```
slrtsetCC ('setup')
```

```
Select your compiler for Simulink Real-Time.

[1] Microsoft Visual Studio 2015 in C:\Program Files (x86)\Microsoft Visual Studio 14.0

[0] None


Compiler: 1

Verify your selection:

Compiler: Microsoft Visual Studio 2015
Location: C:\Program Files (x86)\Microsoft Visual Studio 14.0\

Are these correct [y]/n? y

Done...
```

Set Compiler Specification

```
slrtsetCC ('VisualC','C:\Program Files (x86)\Microsoft Visual Studio 14.0\')
```

Select Visual Studio 2019 Compiler

The `slrtsetCC` function does not support the Visual Studio 2019 compiler. To setup the Visual Studio 2019 compiler for Simulink Real-Time, set the Simulink Real-Time compiler specification to the empty character vector and select the compiler by using `mex -setup`.

```
slrtsetCC ('VisualC','')
mex -setup
```

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 | string scalar

Data Types: `char` | `string`

See Also
`mex` | `slrtgetCC`

External Websites
www.mathworks.com/support/compilers/current_release

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 real-time 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, ___` executes 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 | string scalar

The target name character vector or string scalar is case-sensitive.

Example: `'TargetPC1'`

See Also

Topics

"Troubleshooting in Simulink Real-Time"

Introduced in R2014a

SimulinkRealTime.addTarget

Add target computer interface

Syntax

```
settings_object = SimulinkRealTime.addTarget(target_name)
```

Description

`settings_object = SimulinkRealTime.addTarget(target_name)` adds the definition for a new target computer, represented by the name `target_name`. It returns an object containing the settings for the target computer.

Examples

Add Target `'TargetPC2'` to System

Create a settings object representing target computer `'TargetPC2'`.

```
settings_object = SimulinkRealTime.addTarget('TargetPC2')
```

```
Simulink Real-Time Target Settings

    Name                    : TargetPC2

    TargetRAMSizeMB         : Auto
    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

    TargetScope              : Enabled

    TargetBoot               : BootFloppy
    BootFloppyLocation       :
```

Input Arguments

`target_name` — Name assigned to target computer
character vector | string scalar

Example: `'TargetPC1'`

Data Types: `char` | `string`

Output Arguments

`settings_object` — Settings object representing target computer
`SimulinkRealTime.targetSettings` object

Object containing target computer environment settings.

Data Types: `struct`

See Also

`SimulinkRealTime.getTargetSettings` | `SimulinkRealTime.removeTarget`

Introduced in R2014a

SimulinkRealTime.Application

Represent application files on development computer

Description

Object represents application files on the development computer. You can create it only after the real-time application has been built.

Object provides access to a method that updates external input data for root-level Inport blocks.

Creation

app_object = SimulinkRealTime.Application(*application_name*) creates an object that you can use to manipulate real-time application files on the development computer. You can create it only after the real-time application has been built.

The SimulinkRealTime.Application function accepts these arguments:

- *application_name* — Name of real-time application (character vector or string scalar). For example, 'ex_slrt_inport_osc'.

  This argument is the file name without extension of the .mldatx file that the build produces on the development computer.
- *app_object* — Represent real-time application files on the development computer.

  This argument provides access to methods that manipulate the real-time application files.

This example creates an application object for real-time application ex_slrt_inport_osc.

```
app_object = SimulinkRealTime.Application('ex_slrt_inport_osc');
```

Properties

`ApplicationName` — Name of real-time application

character vector | string scalar

This property is read-only.

Name of real-time application created when application was built.

`UseERAMForLogging` — Mandates use of RAM disk for data logging

`'off'` (default) | `'on'` | character vector | string scalar

When `'on'`, this parameter requires that the real-time application uses the RAM disk (ERAM) for data logging, even when the target computer has a hard drive.

The change takes effect on subsequent downloads of the real-time application. If you rebuild the real-time application, the setting is lost. To make the setting permanent, set the model parameter in the Command Window:

```
set_param(model_name, 'UseERAMForLogging", 'on')
```

The model parameter is hidden.

`UserData` — Add user data to real-time application

`[]` (default) | character vector | numeric vector | cell array

You can assign arbitrary vector data to the **UserData** field. You can access this data only from the development computer.

Example: `{'This string', 10}`

Object Functions

SimulinkRealTime.Application.-updateRootLevelInportData	Replace external input data in a real-time application with new input data

Examples

Update Root-Level Inport Data

Waveform data was originally a square wave. Change it to a sine wave.

Change inport waveform data from a square wave to sine wave.

```
waveform = sinewave;
```

Create an application object.

```
app_object = SimulinkRealTime.Application('ex_slrt_inport_osc');
```

Update inport data.

```
updateRootLevelInportData(app_object)
```

Download the updated inport data to the default target computer.

```
tg = slrt;
load(tg, 'ex_slrt_inport_osc');
```

See Also

Topics

“Define and Update Inport Data”
“Define and Update Inport Data with MATLAB Language”

Introduced in R2017a

SimulinkRealTime.Application.updateRootLevelInportData

Replace external input data in a real-time application with new input data

Syntax

```
updateRootLevelInportData(app_object)
```

Description

`updateRootLevelInportData(app_object)` updates

Examples

Update Inport Data with Application Object

Create an application object for real-time application `ex_slrt_inport_osc`. Use it to update inport data.

Change inport waveform data from a square wave to sine wave.

```
waveform = sinewave;
```

Create an application object.

```
app_object = SimulinkRealTime.Application('ex_slrt_inport_osc');
```

Update inport data.

```
updateRootLevelInportData(app_object)
```

Download the updated inport data to the default target computer.

```
tg = slrt;
load(tg, 'ex_slrt_inport_osc');
```

Input Arguments

app_object — Represent real-time application files on the development computer

object

Provides access to methods that manipulate the real-time application files.

See Also

`SimulinkRealTime.Application` | `SimulinkRealTime.target`

Topics

“Define and Update Inport Data”
“Define and Update Inport Data with MATLAB Language”

Introduced in R2017a

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.

Note: The `SimulinkRealTime.fileSystem` object will be removed in a future release. See the release note for file system commands to use instead. These commands use the `SimulinkRealTime.openFTP` function and the functions for the MATLAB `ftp` object.

`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 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 | string scalar | object

If the argument is a character vector or string scalar, 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` | `string` | `struct`

`file_name` — Name of a file on the target computer

file name character vector or string scalar | full path name character vector or string scalar

If the argument is a file name, the file must be in the current folder on the target computer, as indicated by the function `SimulinkRealTime.fileSystem.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` | `string`

See Also

`SimulinkRealTime.copyFileToTarget` | `cd` | `dir` | `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.

Note: The `SimulinkRealTime.fileSystem` object will be removed in a future release. See the release note for file system commands to use instead. These commands use the `SimulinkRealTime.openFTP` function and the functions for the MATLAB `ftp` object.

`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 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 | string scalar | object

If the argument is a character vector or string scalar, the argument 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` | `string` | `struct`

`file_name` — Name of a file in the current folder on the development computer

file name character vector or string scalar | full path name character vector or string scalar

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 `SimulinkRealTime.fileSystem.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` | `string`

See Also

`SimulinkRealTime.copyFileToHost` | `cd` | `dir` | `pwd`

Introduced in R2014a

SimulinkRealTime.crashInfo

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.

For Network, DOS Loader, and Removable Disk boot modes, twenty seconds after a fatal error, the target computer restarts itself and saves the crash data on the target computer hard drive.

For Stand Alone boot mode, twenty seconds after a fatal error, the target computer restarts itself, then you can use a manual standalone boot to make the target computer save the crash data on the target computer hard drive. For more information, see “Standalone Boot Method”.

When the computer is running again, you can call the `display` function from the development computer to retrieve the crash data.

Creation

`cinfo_object` `= SimulinkRealTime.crashInfo(`*`target_name`*`)` and `cinfo_object = SimulinkRealTime.crashInfo(`*`settings_object`*`)` create and return a crash information object.

If a CPU exception occurred, it calls `update` and `display` to print the crash information.

If a CPU exception did not occur, `SimulinkRealTime.crashInfo` produces an error message.

The `SimulinkRealTime.crashInfo` function accepts these arguments:

- *`target_name`* — Name assigned to target computer (character vector or string scalar). For example, `'TargetPC1'`.
- *`target_object`* — Object representing target computer that had the CPU exception.
- *`settings_object`* — Settings object representing target computer

  Object containing target computer environment settings.
- *`cinfo_object`* — Object representing crash information

  Object that provides properties and functions for accessing crash information.

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 | string scalar

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

display Display crash information
update Update crash information object

Examples

Get Crash Information After CPU Exception

Create a `SimulinkRealTime.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 `SimulinkRealTime.crashInfo` object.

```
cinfo_object = SimulinkRealTime.crashInfo('TargetPC1')
```

```
Crash information object saved as C:\Users\AppData\Local\...
    Temp\SLRTCrashInfo_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
```

For technical support, send the `SLRTCrashInfo*.mat` file to MathWorks Support (`www.mathworks.com/support`).

Get Crash Information by Target Computer Name

Create a `SimulinkRealTime.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 `SimulinkRealTime.crashInfo` object.

```
cinfo_object = SimulinkRealTime.crashInfo('TargetPC1')
```

```
Crash information object saved as C:\Users\AppData\Local\...
    Temp\SLRTCrashInfo_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
```

Get Crash Information by Target Object

Create a `simulinkRealTime.crashInfo` object by target object and display crash information.

Create and display a `SimulinkRealTime.crashInfo` object for the default target computer.

```
target_object = slrt;
cinfo_object = SimulinkRealTime.crashInfo(target_object)
```

```
Crash information object saved as C:\Users\AppData\Local\...
    Temp\SLRTCrashInfo_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
```

Get Crash Information by Settings Object

Create a `SimulinkRealTime.crashInfo` object by settings object and display crash information.

Create and display a `SimulinkRealTime.crashInfo` object.

```
settings_object = SimulinkRealTime.getTargetSettings('TargetPC1');
cinfo_object = SimulinkRealTime.crashInfo(settings_object)

Crash information object saved as C:\Users\AppData\Local\...
    Temp\SLRTCrashInfo_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
```

See Also

`SimulinkRealTime.getSupportInfo`

Topics

“Find Simulink Real-Time Support”
“Troubleshoot Missing or Unreadable Crash Information”

External Websites

www.mathworks.com/support

Introduced in R2016b

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.

See Also

`SimulinkRealTime.crashInfo`

Topics

“Troubleshoot Missing or Unreadable Crash Information”

Introduced in R2016b

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 `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 `SimulinkRealTime.crashInfo` object.

```
update(cinfo_object);
```

```
Crash information object saved as C:\Users\AppData\Local\...
    Temp\SLRTCrashInfo_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_sfun.c, line 106
Function:          mdlOutputs
Message:           Found in model code
```

Input Arguments

`cinfo_object` — Object representing the crash information

structure

Object that provides properties and functions for accessing crash information.

See Also

`SimulinkRealTime.crashInfo` | `display`

Topics

“Troubleshoot Missing or Unreadable Crash Information”

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 Run-Time 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.

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`

See Also

`SimulinkRealTime.addTarget` | `SimulinkRealTime.getTargetSettings` | `SimulinkRealTime.target` | Target Settings | `slrt`

Topics

“Target Computer Boot Methods”
“Command-Line Target Computer Boot Methods”

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. This function generates the file `slrtinfo.m` in the current folder.

```
summary = SimulinkRealTime.getSupportInfo
```

Target Computer and Model Information

Get diagnostic information about a functioning target computer and real-time application. This function generates the files `slrtinfo.m` and `SLRTDebug.m` in the current folder.

```
summary = SimulinkRealTime.getSupportInfo('testmodel')
```

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. This function generates the files `slrtinfo.m` and `SLRTDebug.m` in the current folder. It generates the file `SLRTCrashInfo*.mat` on the development computer hard drive.

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')
```

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 with these fields:

- `antivirus` — The antivirus software of the development computer
- `compiler` — The compiler name installed on the development computer
- `compilerPath` — The compiler installation folder location
- `consoleLog` — The contents of the top entry in the target computer console log
- `consoleLogStruct` — A struct that contains `sequence`, `attrib`, and `text` fields for each console log entry
- `cpuInfo` — The CPU information for the target computer
- `crashInfo` — The information about the fatal error if the target computer had a fatal error, and otherwise does not appear
- `crashStatus` — The value is 1 if the target computer had a fatal error, and otherwise 0
- `date` — The date and time
- `dosTargetPing` — The real-time kernel communication information
- `dosversion` — The Windows version of the development computer
- `getPCIInfo` — The PCI bus devices information for the target computer
- `hostname` — The development computer name
- `kernelInfo` — The real-time kernel information
- `kernelNames` — The real-time kernel information
- `kernelTime` — The real-time kernel information
- `matlabroot` — The MATLAB installation folder location
- `mexConfig` — An array with these fields for each Mex compiler: `Name`, `Manufacturer`, `Language`, `Version`, `Location`, `ShortName`, `Priority`, `Details`, `LinkerName`, `LinkerVersion`, `MexOpt`

- `mexVerbose` — The status of the search for available Mex compilers installed on the development computer
- `path` — The MATLABpath
- `pwd` — The current working folder
- `selfPing` — The real-time kernel communication information
- `slrtroot` — The Simulink Real-Time installation folder location
- `speedgoatver` — The Speedgoat driver library version information if library installed, and otherwise does not appear
- `targetSettings` — The target computer settings
- `ver` — A struct that contains the `Name`, `Version`, `Release`, `Date`, and `Licenses` of the installed MathWorks products

See Also

`SimulinkRealTime.crashInfo`

Topics

“Find Simulink Real-Time Support”

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_objects)
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_objects)` 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
    LegacyMultiCoreConfig   : on
    USBSupport              : on
    ShowHardware            : off
    EthernetIndex           : 0

    TcpIpTargetAddress      : 10.10.10.15
    TcpIpTargetPort         : 22222
    TcpIpSubNetMask         : 255.255.255.0
    TcpIpGateway            : 10.10.10.100
    TcpIpTargetDriver       : Auto
    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
    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 | string scalar

The name character vector or string scalar of a target computer.

Example: `'TargetPC1'`

Data Types: `char` | `string`

`all_objects` — Selects all objects list
`'-all'` | character vector | string scalar

The `-all` argument as character vector or string scalar. If selected, the function displays a list of environment objects representing all defined target computers.

Example: `'-all'`

Data Types: `char` | `string`

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

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` when the development computer and the default target computer can communicate using the settings for the default computer. Otherwise, it returns `failed`. If a communication channel is open between the development and target computers, the function leaves it open.

`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`. If a communication channel is open between the development and target computers, the function leaves it open.

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`

See Also

Introduced in R2014a

SimulinkRealTime.removeTarget

Remove target computer interface

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 default target computer, the next target object becomes the default target computer. Do not remove the last target computer.

Examples

Remove Target 'TargetPC2' from System

Remove the definitions and settings for `'TargetPC2'` from the system.

```
SimulinkRealTime.removeTarget('TargetPC2')
```

Input Arguments

`target_name` — Name assigned to target computer
character vector | string scalar

Example: `'TargetPC1'`

Data Types: `char` | `string`

See Also

`SimulinkRealTime.addTarget` | `SimulinkRealTime.getTargetSettings`

Introduced in R2014a

SimulinkRealTime.etherCAT.filterNotifications

Display EtherCAT notifications in human-readable format

Syntax

```
SimulinkRealTime.etherCAT.filterNotifications()
SimulinkRealTime.etherCAT.filterNotifications(tlog, olog, suppress)
filtered_values = SimulinkRealTime.etherCAT.filterNotifications(
tlog, olog, suppress)
[filtered_values suppressed_values] =
SimulinkRealTime.etherCAT.filterNotifications(tlog, olog, suppress)
```

Description

`SimulinkRealTime.etherCAT.filterNotifications()` without arguments prints the valid notification values and their text descriptions.

`SimulinkRealTime.etherCAT.filterNotifications(tlog, olog, suppress)` extracts from `olog` the notification values that come from the EtherCAT Get Notifications block, and from `tlog`, the times at which these values occurred.

If the `suppress` vector is nonempty, the function removes from the output list the notification values that appear in the vector. For each code listed in the `suppress` vector, the function prints the total number of occurrences and the time range over which they occurred.

When you are debugging EtherCAT® issues, use this function. You must have advanced knowledge about EtherCAT functionality.

`filtered_values = SimulinkRealTime.etherCAT.filterNotifications( tlog, olog, suppress)` returns a structure vector containing the filtered values.

`[filtered_values suppressed_values] = SimulinkRealTime.etherCAT.filterNotifications(tlog, olog, suppress)`

returns a structure vector containing the filtered values and a structure containing a summary of the suppressed values.

Examples

Print Valid Notifications

Print the valid notification values and their text descriptions

```
SimulinkRealTime.etherCAT.filterNotifications
```

```
SimulinkRealTime.EtherCAT.filterNotifications
(     1): State changed
(     2): Cable connected
(     3): Scanbus finished
(     4): Distributed clocks initialized
(     5): DC slave synchronization deviation received
(     8): DCL initialized
(     9): DCM inSync
(    21): Successful slave state transition.
(   100): Queue raw command response notification
( 65537): Cyclic command: Working count error
( 65538): Master init command: Working count error
( 65539): Slave init command: Working count error
( 65540): EOE mbox receive: Working count error (deprecated)
( 65541): COE mbox receive: Working count error (deprecated)
( 65542): FOE mbox receive: Working count error (deprecated)
( 65543): EOE mbox send: Working count error
( 65544): COE mbox send: Working count error
( 65545): FOE mbox send: Working count error
( 65546): Frame response error: No response
( 65547): Slave init command: No response
( 65548): Master init command: No response
( 65550): Timeout when waiting for mailbox init command response
( 65551): Cyclic command: Not all slaves in op state
( 65552): Ethernet link (cable) not connected
( 65554): Redundancy: Line break detected
( 65555): Cyclic command: A slave is in error state
( 65556): Slave error status change
( 65557): Station address lost (or slave missing) - FPRD to ...
          AL_STATUS failed
( 65558): SOE mbox receive: Working count error (deprecated)
```

```
( 65559): SOE mbox send: Working count error
( 65560): SOE mbox write responded with an error
( 65561): COE mbox SDO abort
( 65562): Client registration dropped, possibly call to ...
          ecatConfigureMaster by other thread (RAS)
( 65563): Redundancy: Line is repaired
( 65564): FOE mbox abort
( 65565): Invalid mail box data received
( 65566): PDI watchdog expired on slave, thrown by IST
( 65567): Slave not supported (if redundancy is activated and ...
          slave doesn't fully support autoclose
( 65568): Slave in unexpected state
( 65569): All slave devices are in operational state
( 65570): VOE mbox send: Working count error
( 65571): EEPROM checksum error detected
( 65572): Crossed lines detected
( 65573): Junction redundancy change
(196610): ScanBus mismatch
(196611): ScanBus mismatch. A duplicate HC group was detected
(262146): HC enhance detect all groups done
(262147): HC probe all groups done
(262148): HC topology change done
(262149): Slave disappears
(262150): Slave appears
```

Print Notifications from Normal Operation

Print the notifications that appear during normal operation.

In this example, the output signals from the EtherCAT Get Notifications block are at signal indexes `1:21`. Pass in an empty `suppress` vector. Notifications are subject to change.

```
tlog = tg.TimeLog;
olog = tg.OutputLog(:, 1:21);
SimulinkRealTime.etherCAT.filterNotifications(tlog, olog, [])
```

```
 Time        Notification
0.039000 (     3) Scanbus Finished
0.054000 (     1) State Change
0.085000 (     1) State Change
0.281000 (     4) Distributed clocks initialized
3.280000 (     9) DCM inSync
```

```
3.491000 (     5) DC Slave Synchronization deviation received
3.494000 (     1) State Change
3.500000 (     1) State Change
         ( 65569) All slave devices are in operational state
```

Print Filtered Notifications from Normal Operation

Filter and print the notifications that appear during normal operation. Filter notification `( 1) State Change`.

In this example, the output signals from the EtherCAT Get Notifications block are at signal indexes `1:21`. Notifications are subject to change.

```
tlog = tg.TimeLog;
olog = tg.OutputLog(:, 1:21);
SimulinkRealTime.etherCAT.filterNotifications( tlog, olog, [1]);
```

```
 Time        Notification
0.039000 (     3) Scanbus Finished
0.281000 (     4) Distributed clocks initialized
3.280000 (     9) DCM inSync
3.491000 (     5) DC Slave Synchronization deviation received
         ( 65569) All slave devices are in operational state

1: 4 times [0.054000 : 3.500000]
State Change
```

Return Notifications from Normal Operation

Return as a vector the notifications that appear during normal operation.

In this example, the output signals from the EtherCAT Get Notifications block are at signal indexes 1–21. Pass in an empty `suppress` vector.

```
tlog = tg.TimeLog;
olog = tg.OutputLog(:, 1:21);
filtered_values = ...
    SimulinkRealTime.etherCAT.filterNotifications(tlog, olog, [])
```

```
filtered)values =
  1×7 struct array with fields:
```

```
    time
    code
    notifystring
```

Return Filtered Notifications from Normal Operation

Filter and return the notifications that appear during normal operation. Filter notification `( 1) State Change`.

In this example, the output signals from the EtherCAT Get Notifications block are at signal indexes `1:21`.

```
tlog = tg.TimeLog;
olog = tg.OutputLog(:, 1:21);
[filtered_values suppressed_values] = ...
    SimulinkRealTime.etherCAT.filterNotifications(tlog, olog, [1])
```

```
filtered_values =
  1×3 struct array with fields:
    time
    code
    notifystring

suppressed_values =
  struct with fields:

      val: 1
    first: 0.013000000000000
     last: 1.216500000000000
    count: 4
```

Input Arguments

`tlog` — Time log on target computer

vector

Read `tg.TimeLog` from the target computer.

Example: `tg.TimeLog`

Data Types: `double`

`olog` — Output log on target computer
matrix

Read `tg.OutputLog` from the target computer.

Example: `tg.OutputLog`

Data Types: `double`

`suppress` — List of notification codes to omit from the line-by-line report
vector

For each code, the function reports the total number of occurrences and the time range over which they occurred. If you do not want to suppress notification codes, pass in an empty vector (`[]`).

Example: `65546`

Example: []

Data Types: `double`

Output Arguments

`filtered_values` — Return filtered values as a structure vector
vector

Each element of `filtered_values` is a structure containing:

- `time` (double) — Timestamp of notify code
- `code` (double) — Notify code
- `notifystring` (character vector) — Text description

`suppressed_values` — Return suppressed codes as a structure vector
vector

Each element of `suppressed_values` is a structure containing:

- `val` (double) — Notify code
- `first` (double) — Timestamp of first occurrence
- `last` (double) — Timestamp of last occurrence

- `count` (double) — Number of instances found

Tips

- To retrieve the notifications, in the Command Window, read `tg.OutputLog` and `tg.TimeLog` from the target computer.
- Determine which `tg.OutputLog` columns to pass into `SimulinkRealTime.etherCAT.filterNotifications`.
  - If you connected the EtherCAT Get Notifications block to the first Outport block, the 21 notification signals appear in columns `1:21` of the `tg.OutputLog` matrix.
  - To determine which columns of `tg.OutputLog` come from the EtherCAT Get Notifications block, set `tg.ShowSignals` to `'on'`. From the resulting information, determine the relevant columns.
- Common error conditions, such as an unplugged Ethernet cable, can cause thousands of unwanted notifications that hide useful notifications. To filter unwanted notifications, use the `suppress` vector.
- The EtherCAT Get Notifications block can quickly increase the size of the output log. In the **Simulink Real-Time Options** pane, if the value of **Signal logging data buffer size in doubles** is too small, the log wraps and overwrites the oldest data, which can contain important diagnostic information.

See Also

EtherCAT Get Notifications

Introduced in R2017a

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.

Data sometimes appears in row-major format (a row, not a column, refers to a time step). In such cases, transpose the variable and pass the result 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)
```

Configure 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));
```

Configure 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 | string scalar

The data file contains columns of data to be output to the model.

Example: `'myfile'`

Data Types: `char` | `string`

`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`

See Also

Introduced in R2014a

SimulinkRealTime.utils.getConsoleLog

Retrieve the log that the kernel writes to the target computer screen

Syntax

```
console_log = SimulinkRealTime.utils.getConsoleLog(target_name,
sequence_number)
console_log = SimulinkRealTime.utils.getConsoleLog(target_object,
sequence_number)
```

Description

`console_log = SimulinkRealTime.utils.getConsoleLog(target_name, sequence_number)` retrieves the log from the target computer, `target_name`, for lines equal to or greater than the sequence number. If the `target_name` is omitted, the function returns the console log of the default target computer (for example, `'TargetPC1'`). If the `sequence_number` is omitted, the function returns lines from the default sequence (for example, `0`). If no lines have such a sequence number, the function returns an empty array.

The `console_log` is an array of structures, one per line of text in the console. Each structure has the following fields:

- `sequence:` Absolute sequence number of the console line, after target reboot.
- `attrib:` The type of message the line represents:
  - 0 - normal messages
  - 1 - warning messages
  - 2 - error messages
- `text:` The text of the console line. The text does not include a line termination character (for example, `'\n'`).

`console_log = SimulinkRealTime.utils.getConsoleLog(target_object, sequence_number)` retrieves the log from the target computer identified by the object, `target_object`.

Examples

Get Log for Default Target Computer and Default Sequence Number

Get the console log for the default target computer starting from sequence number `0`.

```
console_log = SimulinkRealTime.utils.getConsoleLog
% if the default target computer is TargetPC1,
% this function is equivalent to:
% console_log = SimulinkRealTime.utils.getConsoleLog('TargetPC1',0)
```

```
console_log =

1×13 struct array with fields:

sequence
attrib
text
```

```
console_log(1)
```

```
ans =

struct with fields:

sequence: 0
attrib: 0
text: 'Starting up with 1 CPU'
```

Get Log for Target Computer and Sequence Number

Get the console log for a target computer selected with a target object, starting from a specific sequence number.

```
tg = slrt;
console_log = SimulinkRealTime.utils.getConsoleLog(tg,5)
```

```
console_log =

1×8 struct array with fields:

sequence
```

```
attrib
text
```

```
console_log(1)
```

```
ans =

struct with fields:

sequence: 5
attrib: 0
text: 'Download finished'
```

Input Arguments

`target_name` — Name assigned to target computer

character vector | string scalar

Example: `'TargetPC1'`

Data Types: `char` | `string`

`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`

`sequence_number` — Target computer console log sequence number

`0` (default) | integer

The target computer log sequence number indicates the line number from the beginning of last reboot, starting from `0`. This number does not necessarily match the line number displayed on the screen or observed at the buffer.

Example: `5`

Output Arguments

`console_log` — Lines printed to target computer screen
vector of struct

The function returns the console log as a vector of structures consisting of the fields: `sequence`, `attrib`, and `text`.

See Also

Topics

“Control Real-Time Application at Target Computer Command Line”

Introduced in R2017a

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 `fread` method.

Examples

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))
```

Store, Convert, 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

Scope | `SimulinkRealTime.copyFileToHost` | `SimulinkRealTime.fileSystem`

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`

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,
build_mode)
```

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, build_mode)` 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 | string scalar

The model name character vector or string scalar.

Example: `'xpcosc'`

Data Types: `char` | `string`

`build_mode` — Select clean up after build

`'-cleanup'` | character vector | string scalar

The `-cleanup` argument as character vector or string scalar. If selected, when execution is complete, the function deletes the build files.

Example: `'-cleanup'`

Data Types: `char` | `string`

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.

See Also

Topics

“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`

`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

SimulinkRealTime.utils.TETMonitor.open

Display TET monitor

Syntax

```
SimulinkRealTime.utils.TETMonitor.open
```

Description

`SimulinkRealTime.utils.TETMonitor.open` opens the task execution time (TET) monitor in the MATLAB session that is available for all Simulink Real-Time target objects. You can open the TET monitor at any time. Depending on the current state of connected target computers, the monitor displays TET data for each real-time application task or displays a message indicating the issue that prevents TET data display. Changes to the target computer state are updated in the TET monitor. The monitor displays these target states:

- **Waiting**: Displays no TET data because no real-time application is executing.
- **Disabled**: Displays no TET data because a real-time application is executing with TET monitoring disabled.
- **Active**: Actively streams and displays TET data as a real-time application is executing.

Examples

Open TET Monitor and View Status

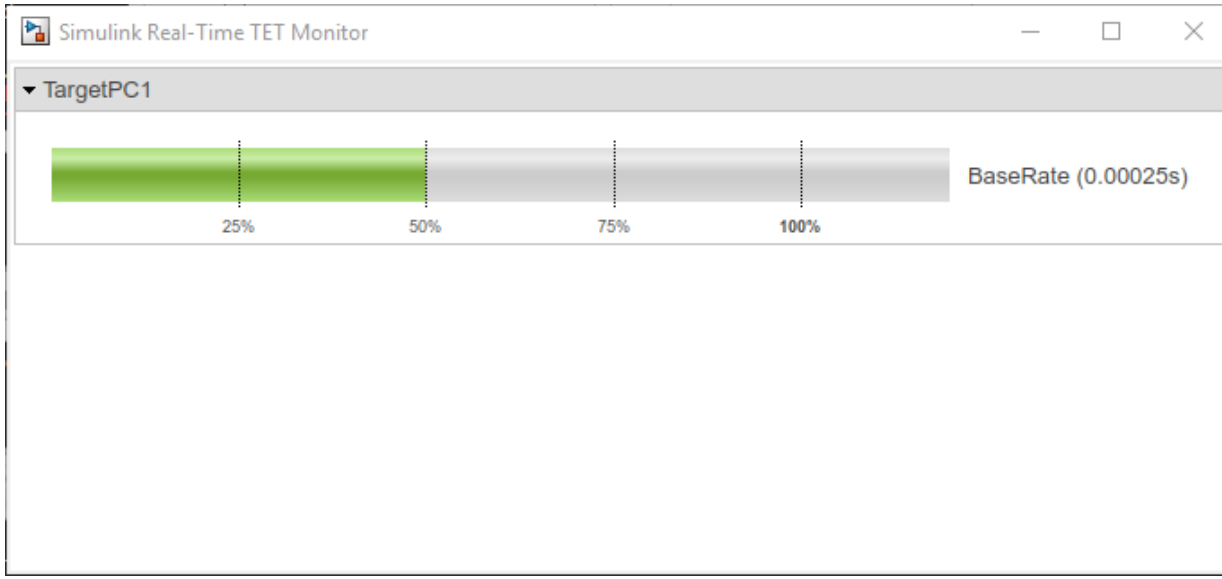
In the “Inspect Simulink® Real-Time™ Data with Simulation Data Inspector” example, use these added steps to display the TET monitor for the example.

After you open the `xpcosc` model, open the **Configuration Parameters**. In the **Real-Time** tab, click **Hardware Settings**. Select the **Code Generation > Simulink Real-Time Options > Data logging options > Monitor Task Execution Time** option.

Before you build the model and download it to the target computer, open the TET monitor. In the **Real-Time** tab, click **TET Monitor**. Or, in the Command Window, enter:

```
SimulinkRealTime.utils.TETMonitor.open
```

When you run the real-time application, the TET monitor displays status.



See Also

Task Execution Time | `xPCGetTETLog`

Topics

“Inspect Simulink® Real-Time™ Data with Simulation Data Inspector”

Introduced in R2018a

Target Settings Properties

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.

Note

- Support for using ISA bus Ethernet cards to communicate between the development and target computers has ceased to function. Use PCI bus or USB bus Ethernet cards instead.
- The `NonPentiumSupport` property has no function.
- In a future release, the `SecondaryIDE` target setting will be read-only and set to `'off'`.
- The `MulticoreSupport` target setting is read-only and set to `'on'`.
- The `MaxModelSize` property has no function.
- The **RAM size** check box has been removed from Simulink Real-Time Explorer. The `TargetRAMSizeMB` property value continues to function.

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
    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

    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**.

Properties

Host-to-Target Communication

`TcpIpGateway` — IP address for gateway to Ethernet link

`'255.255.255.255'` (default) | `'xxx.xxx.xxx.xxx'` | character vector | string scalar

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'` | character vector | string scalar

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'` | character vector | string scalar

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) | `'USB'` | character vector | string scalar

This property determines the bus type of your target computer. You do not need to define a bus type for your development computer.

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) | `'I210'` | `'I217'` | `'I8254x'` | `'I82559'` | `'R8139'` | `'R8168'` | `'USBAX172'` | `'USBAX772'` | `'X540'` | character vector | string scalar

If the target computer contains only one supported Ethernet card, use the default value (`'Auto'`).

If you are using bus type `'USB'`, use `'USBAX172'` or `'USBAX772'`.

In the Simulink Real-Time Explorer **Target driver** list, select one of `INTEL_I210`, `INTEL_I217`, `INTEL_I8254x`, `INTEL_I82559`, `INTEL_X540`, `R8139`, `R8168`, `USBAX172`, `USBAX772`, or `Auto`.

Example: `env_object.TcpIpTargetDriver = 'USBAX172'`

`TcpIpTargetPort` — Ethernet port on target computer

`'22222'`. (default) | `'xxxxx'` | character vector | string scalar

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'` | character vector | string scalar

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'` | character vector | string scalar

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'`

`Name` — Target computer name
`'TargetPC`*N*`'` (default) | character vector | string scalar

When you create a target settings object, the software assigns it a name of the form `'TargetPC`*N+1*`'`. `'TargetPC`*N*`'` 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'`

`ShowHardware` — Display Ethernet card information for target computer
`'off'` (default) | `'on'` | character vector | string scalar

To display Ethernet card information on the target monitor, set `ShowHardware` to `'on'` and then start the target computer. The target computer monitor displays the index, bus, slot, function, and target driver for each Ethernet card.

With `ShowHardware` set, after the kernel starts, the development computer cannot communicate with the target computer. When you have gathered your information, 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*`'` | character vector | string scalar

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 4 GB.

Example: `env_object.ShowHardware = '2000'`

`TargetScope` — Display scope information graphically

`'Enabled'` (default) | `'Disabled'` | character vector | string scalar

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 use target scopes 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'` | character vector | string scalar

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 | string scalar

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 | string scalar

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'` | character vector | string scalar

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'` | character vector | string scalar

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`

Topics

“PCI Bus Ethernet Setup”
“Target Computer Settings”
“Target Computer Boot Methods”

Introduced in R2014a

SimulinkRealTime.targetSettings.setAsDefaultTarget

Set specific target computer as default target computer

Syntax

```
setAsDefaultTarget(settings_object)
```

Description

`setAsDefaultTarget(settings_object)` marks the target computer represented by the target settings object as the default target computer.

Examples

Make Target Computer 'TargetPC1' the Default

Get the target settings object for target computer `'TargetPC1'` and make that target computer the default target computer.

```
settings_object = SimulinkRealTime.getTargetSettings('TargetPC1');
setAsDefaultTarget(settings_object)
```

Input Arguments

`settings_object` — Settings object representing target computer
`SimulinkRealTime.targetSettings` object

Object containing target computer environment settings.

Data Types: `struct`

See Also

`SimulinkRealTime.addTarget` | `SimulinkRealTime.getTargetSettings`

Introduced in R2014a

SimulinkRealTime.fileSystem

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 at most 128 files open on the target computer at the same time.
- The largest single file that you can create on the target computer 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 in the operating system on the target computer can have a maximum of 260 characters. If the file name is longer than eight-dot-three format (eight character file name, period, three character extension), the operating system represents the file name in truncated form (for example, six characters followed by `'~1'`). MATLAB commands can access the file using the fully qualified file name or the truncated representation of the name. Some block parameters, such as the Scope block `filename` parameter, require 8.3 format for the file name.
- Do not write data to the `private` folder on your target computer. It is reserved for Simulink Real-Time internal use.

Note: The `SimulinkRealTime.fileSystem` object will be removed in a future release. See the release note for file system commands to use instead. These commands use the `SimulinkRealTime.openFTP` function and the functions for the MATLAB `ftp` object.

Creation

`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.

Note: The `SimulinkRealTime.fileSystem` object will be removed in a future release. See the release note for file system commands to use instead. These commands use the `SimulinkRealTime.openFTP` function and the functions for the MATLAB `ftp` object.

`filesys_object = SimulinkRealTime.fileSystem(target_object)` constructs and returns the file system object corresponding to the target computer that is accessible by `target_object`.

The `SimulinkRealTime.fileSystem` function accepts these arguments:

- *`target_object`* — Object representing target computer

  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.
- *`filesys_object`* — Object representing the target computer file system

  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.

Properties

No properties.

Object Functions

cd	Change folder on target computer
dir	List contents of folder on target computer
diskinfo	Target computer drive information
diskspace	Return the free space and total space on the drive, in bytes
fclose	Close target computer file
fileinfo	Target computer file information
filetable	Information about open files in target computer file system
fopen	Open target computer file for reading and writing
fread	Read open target computer file

fwrite	Write binary data to open target computer file
getfilesize	Size of file on target computer
mkdir	Create folder on target computer
pwd	Path to currently active folder on target computer
removefile	Remove file from target computer
rename	Rename a file or folder in the target computer disk drive
rmdir	Remove empty folder from target computer
selectdrive	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
```

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
```

See Also

`slrt`

Introduced in R2014a

cd

Change folder on target computer

Syntax

```
cd(filesys_object, folder_name)
```

Description

`cd(filesys_object, folder_name)` changes the currently active folder on the target computer. Prints an error if the destination folder does not exist.

Note: The `SimulinkRealTime.fileSystem` object will be removed in a future release. See the release note for file system commands to use instead. These commands use the `SimulinkRealTime.openFTP` function and the functions for the MATLAB `ftp` object.

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`

`folder_name` — Name of a folder on the target computer

character vector | string scalar

If you omit the drive letter, the command assumes that the folder path is relative to the default drive.

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

Example: `logs`

Data Types: `char`

See Also

`SimulinkRealTime.fileSystem` | `mkdir` | `slrt`

Introduced in R2014a

dir

List contents of 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.

Note: The `SimulinkRealTime.fileSystem` object will be removed in a future release. See the release note for file system commands to use instead. These commands use the `SimulinkRealTime.openFTP` function and the functions for the MATLAB `ftp` object.

`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 | string scalar

If you omit the drive letter, the command assumes that the folder path is relative to the default drive.

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

Example: `logs`

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 filer 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

`SimulinkRealTime.fileSystem` | `mkdir` | `slrt`

Introduced in R2014a

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.

Note: The `SimulinkRealTime.fileSystem` object will be removed in a future release. See the release note for file system commands to use instead. These commands use the `SimulinkRealTime.openFTP` function and the functions for the MATLAB `ftp` object.

Examples

Return Configuration Information About Specified Disk

Return configuration information for the target computer `C:\` drive.

```
tg = slrt;
filesys_object = SimulinkRealTime.fileSystem(tg);
disk_info = diskinfo(filesys_object,'C:\')
```

```
disk_info =

  struct with fields:

           DriveLetter: 'C'
                 Label: 'FREEDOS'
              Reserved: '   '
          SerialNumber: -857442364
    FirstPhysicalSector: 63
```

```
           FATType: 32
          FATCount: 2
      MaxDirEntries: 0
     BytesPerSector: 512
  SectorsPerCluster: 64
      TotalClusters: 1831212
        BadClusters: 0
       FreeClusters: 1827626
              Files: 932
         FileChains: 936
         FreeChains: 1
   LargestFreeChain: 1827626
         DriveType: DRIVE_FIXED
```

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 on 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, the drive type, 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` | `slrt`

Introduced in R2014a

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.

Note: The `SimulinkRealTime.fileSystem` object will be removed in a future release. See the release note for file system commands to use instead. These commands use the `SimulinkRealTime.openFTP` function and the functions for the MATLAB `ftp` object.

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 on 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` | `slrt`

Introduced in R2016a

fclose

Close target computer file

Syntax

```
status = fclose(filesys_object,file_id)
```

Description

`status = fclose(filesys_object,file_id)` closes an open file in the target computer file system. `file_id` is the file identifier associated with an open file.

You can have at most 128 files open on the target computer at the same time.

`fclose` does not close standard input, standard output, or standard error.

Note: The `SimulinkRealTime.fileSystem` object will be removed in a future release. See the release note for file system commands to use instead. These commands use the `SimulinkRealTime.openFTP` function and the functions for the MATLAB `ftp` object.

Examples

Open a File for Writing and Close It

Open file `data.dat`, write to it, and close it again.

Open and write file.

```
tg = slrt;
filesys_object = SimulinkRealTime.fileSystem(tg);
file_id = fopen(filesys_object, 'data.dat', 'w');
fwrite(filesys_object, file_id, 'test')
```

Close file.

```
fclose(filesys_object, file_id)

ans =

     0
```

Open a File for Reading and Close It

Open file `data.dat`, read from it, and close it again.

Open and read file.

```
tg = slrt;
filesys_object = SimulinkRealTime.fileSystem(tg);
file_id = fopen(filesys_object, 'data.dat', 'r');
value = fread(filesys_object, file_id);
char(value)

ans =

  1×4 char array

test
```

Close file.

```
fclose(filesys_object, file_id)

ans =

     0
```

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`

`file_id` — Identifier representing a file on the target computer
integer

Pass this value to functions that access files on the target computer.

Example: `h`

Output Arguments

`status` — Indication of whether the file closed properly
`0` | `-1`

If the file closed properly, the value is `0`, otherwise it is `-1`.

See Also

`SimulinkRealTime.fileSystem` | `fopen`

Introduced in R2014a

fileinfo(filesys_obj,file_id)

Target computer file information

Syntax

```
file_info = fileinfo(filesys_object, file_id)
```

Description

`file_info = fileinfo(filesys_object, file_id)` gets file configuration information for the file on the target computer associated with `file_id`.

Note: The `SimulinkRealTime.fileSystem` object will be removed in a future release. See the release note for file system commands to use instead. These commands use the `SimulinkRealTime.openFTP` function and the functions for the MATLAB `ftp` object.

Examples

Get File Information for a File

Open file `data.dat` and read its file information.

```
tg = slrt;
filesys_object = SimulinkRealTime.fileSystem(tg);
file_id = fopen(filesys_object, 'data.dat', 'r');
fileinfo(filesys_object, file_id)
```

```
ans =

  struct with fields:

              FilePos: 0
        AllocatedSize: 32768
        ClusterChains: 1
```

```
    VolumeSerialNumber: -857442364
               FulName: 'C:\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`

`file_id` — Identifier representing a file on the target computer

integer

Pass this value to functions that access files on the target computer.

Example: `h`

Output Arguments

`file_info` — File configuration information

struct

The file information includes the full file name, the amount of space allocated for the file, and technical information for use by a maintenance technician.

See Also

`SimulinkRealTime.fileSystem` | `fopen` | `slrt`

Introduced in R2014a

filetable

Information about open files in target computer file system

Syntax

```
open_file_table = filetable(filesys_object)
```

Description

`open_file_table = filetable(filesys_object)` returns a table of the open files in the target computer file system.

You can have at most 128 files open on the target computer at the same time.

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.

Note: The `SimulinkRealTime.fileSystem` object will be removed in a future release. See the release note for file system commands to use instead. These commands use the `SimulinkRealTime.openFTP` function and the functions for the MATLAB `ftp` object.

Examples

Get the Handle for an Open File

Open a file, get the table containing its file handle, and close it.

Open a file.

```
tg = slrt;
filesys_object = SimulinkRealTime.fileSystem(tg);
file_id = fopen(filesys_object, 'data.dat', 'r');
```

Get the file handle of the file.

```
filetable(filesys_object)
```

```
ans =

  1×186 char array

Index    Handle  Flags     FilePos  Name

-----------------------------------------

    0  03DF0000  R__             0  C:\data.dat
```

Close the file.

```
fclose(filesys_object, hex2dec('03DF0000'))
```

```
ans =

     0
```

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`

Output Arguments

`open_file_table` — Get list of open files and file handles
struct

The file table includes the full file name, the file handle in hexadecimal, and technical information for use by a maintenance technician.

See Also

`SimulinkRealTime.fileSystem` | `hex2dec` | `slrt`

Introduced in R2014a

fopen

Open target computer file for reading and writing

Syntax

```
file_id = fopen(filesys_object, file_name)
file_id = fopen(filesys_object, file_name, permission)
```

Description

`file_id = fopen(filesys_object, file_name)` opens the file of the specified name on the target computer for reading binary data.

There are the following limitations:

- You can have at most 128 files open on the target computer at the same time.
- The largest single file that you can create on the target computer 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 in the operating system on the target computer can have a maximum of 260 characters. If the file name is longer than eight-dot-three format (eight character file name, period, three character extension), the operating system represents the file name in truncated form (for example, six characters followed by `'~1'`). MATLAB commands can access the file using the fully qualified file name or the truncated representation of the name. Some block parameters, such as the Scope block `filename` parameter, require 8.3 format for the file name.
- Do not write data to the `private` folder on your target computer. It is reserved for Simulink Real-Time internal use.

Note: The `SimulinkRealTime.fileSystem` object will be removed in a future release. See the release note for file system commands to use instead. These commands use the `SimulinkRealTime.openFTP` function and the functions for the MATLAB `ftp` object.

`file_id = fopen(filesys_object, file_name, permission)` opens the file of the specified name on the target computer for reading binary data.

Examples

Open a File with Default Permissions

Open file `data.dat` with default permissions, read from it, and close it again.

Open and read file.

```
tg = slrt;
filesys_object = SimulinkRealTime.fileSystem(tg);
file_id = fopen(filesys_object, 'data.dat');
value = fread(filesys_object, file_id);
char(value)
```

```
ans =

  1×4 char array

test
```

Close file.

```
fclose(filesys_object, file_id)
```

```
ans =

     0
```

Open a File for Writing

Open file `data.dat`, write to it, and close it again.

Open and write file.

```
tg = slrt;
filesys_object = SimulinkRealTime.fileSystem(tg);
file_id = fopen(filesys_object, 'data.dat', 'w');
fwrite(filesys_object, file_id, 'test')
```

Close file.

```
fclose(filesys_object, file_id)
```

```
ans =

     0
```

Open a File for Reading

Open file `data.dat`, read from it, and close it again.

Open and read file.

```
tg = slrt;
filesys_object = SimulinkRealTime.fileSystem(tg);
file_id = fopen(filesys_object, 'data.dat', 'r');
value = fread(filesys_object, file_id);
char(value)
```

```
ans =

  1×4 char array

test
```

Close file.

```
fclose(filesys_object, file_id)
```

```
ans =

     0
```

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`

`file_name` — Name of the file that is being opened

character vector | string scalar

The name of the file can be a name relative to the current folder or a fully qualified path. Enclose the file name in single quotation marks.

Example: `'data.dat'`

`permission` — File access permissions under which the file is being opened

`'r'` (default) | `'w'` | `'a'` | `'r+'` | `'w+'` | `'a+'` | character vector | string scalar

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.

Example: `'w'`

Output Arguments

`file_id` — Identifier representing a file on the target computer

integer

Pass this value to functions that access files on the target computer.

Example: `h`

See Also

`SimulinkRealTime.fileSystem` | `fclose` | `slrt`

Introduced in R2014a

fread

Read open target computer file

Syntax

```
data = fread(filesys_object, file_id)
data = fread(filesys_object, file_id, offset, numbytes)
```

Description

`data = fread(filesys_object, 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.

Note: The `SimulinkRealTime.fileSystem` object will be removed in a future release. See the release note for file system commands to use instead. These commands use the `SimulinkRealTime.openFTP` function and the functions for the MATLAB `ftp` object.

`data = fread(filesys_object, file_id, offset, numbytes)` reads `numbytes` bytes from `file_id` starting from position `offset` and writes the block into matrix `data`.

Examples

Open File for Reading

Open file `data.dat`, read from it, and close it again.

Open and read file.

```
tg = slrt;
filesys_object = SimulinkRealTime.fileSystem(tg);
file_id = fopen(filesys_object, 'data.dat', 'r');
```

```
value = fread(filesys_object, file_id);
char(value)
```

```
ans =

  1×4 char array

test
```

Close file.

```
fclose(filesys_object, file_id)
```

```
ans =

     0
```

Open File for Reading *N* Bytes from Offset

Open file `data.dat` at offset `1`, read `3` bytes from it, and close it again.

Open and read file.

```
tg = slrt;
filesys_object = SimulinkRealTime.fileSystem(tg);
file_id = fopen(filesys_object, 'data.dat', 'r');
value = fread(filesys_object, file_id, 1, 3);
char(value)
```

```
ans =

  1×4 char array

est
```

Close file.

```
fclose(filesys_object, file_id)
```

```
ans =

     0
```

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`

`file_id` — Identifier representing a file on the target computer

integer

Pass this value to functions that access files on the target computer.

Example: `h`

`numbytes` — Maximum number of bytes that `fread` can read

all (default) | integer

`offset` — Position from the beginning of file that `fread` starts to read

`0` (default) | integer

Output Arguments

`data` — Matrix containing the binary data that was read

matrix

To get a count of the total number of bytes read into `data`, call the `length` function. 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.

See Also

`SimulinkRealTime.fileSystem` | `fwrite` | `slrt`

Introduced in R2014a

fwrite

Write binary data to open target computer file

Syntax

```
fwrite(filesys_object, file_id, data)
```

Description

`fwrite(filesys_object, file_id, data)` writes the elements of matrix `data` to the file identified by `file_id`. The `file_id` argument is the file identifier associated with an open file. `fwrite` requires that the file is open with write permission.

Note: The `SimulinkRealTime.fileSystem` object will be removed in a future release. See the release note for file system commands to use instead. These commands use the `SimulinkRealTime.openFTP` function and the functions for the MATLAB `ftp` object.

Examples

Write a Magic Number Matrix to a File

Open `magic.dat` for writing, write it, close it, and read it back.

Open `magic.dat` for writing.

```
tg = slrt;
filesys_object = SimulinkRealTime.fileSystem(tg);
file_id = fopen(filesys_object, 'magic.dat', 'w');
```

Create and write a magic square.

```
msquare = magic(5)
fwrite(filesys_object, file_id, msquare);
```

```
msquare =
```

```
    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9
```

Close the file.

```
fclose(filesys_object, file_id)
```

```
ans =

     0
```

Reopen the file for reading and read it.

```
file_id = fopen(filesys_object, 'magic.dat', 'r');
value = fread(filesys_object, file_id)
```

```
value =

  1×25 uint8 row vector

  Columns 1 through 10

   17   23    4   10   11   24    5    6   12   18

  Columns 11 through 20

    1    7   13   19   25    8   14   20   21    2

  Columns 21 through 25

   15   16   22    3    9
```

Close the file.

```
fclose(filesys_object, file_id)
```

```
ans =

     0
```

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`

`file_id` — Identifier representing a file on the target computer

integer

Pass this value to functions that access files on the target computer.

Example: `h`

`data` — Matrix containing the binary data that is written

matrix

The data is written to the file in column order.

Example: `'test'`

See Also

`SimulinkRealTime.fileSystem` | `fread` | `slrt`

Introduced in R2014a

getfilesize

Size of file on target computer

Syntax

```
file_size = getfilesize(filesys_object, file_id)
```

Description

`file_size = getfilesize(filesys_object, file_id)` returns the size (in bytes) of the file identified by the `file_id` file identifier on the target computer file system.

Note: The `SimulinkRealTime.fileSystem` object will be removed in a future release. See the release note for file system commands to use instead. These commands use the `SimulinkRealTime.openFTP` function and the functions for the MATLAB `ftp` object.

Examples

Read the Size of a File

Open file `data.dat`, read its file size, and close it again.

Get the file system and open the file

```
tg = slrt;
filesys_object = SimulinkRealTime.fileSystem(tg);
file_id = fopen(filesys_object, 'data.dat');
```

Read the file size.

```
file_size = getfilesize(filesys_object,file_id)
```

```
file_size =

        4512
```

Close the file.

```
fclose(filesys_object, file_id);
```

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`

`file_id` — Identifier representing a file on the target computer

integer

Pass this value to functions that access files on the target computer.

Example: `h`

Output Arguments

`file_size` — Number of bytes in the file

integer

This value is the value printed by the `dir` command.

See Also

`SimulinkRealTime.fileSystem` | `fopen` | `slrt`

Introduced in R2014a

mkdir

Create folder on target computer

Syntax

```
mkdir(filesys_object,folder_name)
```

Description

`mkdir(filesys_object,folder_name)` makes a new folder in the current folder on the target computer file system.

Note: The `SimulinkRealTime.fileSystem` object will be removed in a future release. See the release note for file system commands to use instead. These commands use the `SimulinkRealTime.openFTP` function and the functions for the MATLAB `ftp` object.

Examples

Create a Folder

Create a folder `logs` in the target computer file system.

```
tg = slrt;
filesys_object = SimulinkRealTime.fileSystem(tg);
mkdir(filesys_object,'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`

`folder_name` — Name of a folder on the target computer
character vector | string scalar

If you omit the drive letter, the command assumes that the folder path is relative to the default drive.

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

Example: `logs`

Data Types: `char`

See Also

`SimulinkRealTime.fileSystem` | `rmdir` | `slrt`

Introduced in R2014a

pwd

Path to currently active folder on target computer

Syntax

```
active_folder = pwd(filesys_object)
```

Description

`active_folder = pwd(filesys_object)` returns the path to the currently active folder on the target computer. Unless `cd(filesys_object, folder_name)` has been called, the currently active folder is the top folder of the boot drive, usually `C:\`.

Note: The `SimulinkRealTime.fileSystem` object will be removed in a future release. See the release note for file system commands to use instead. These commands use the `SimulinkRealTime.openFTP` function and the functions for the MATLAB `ftp` object.

Examples

Return Current Folder

Return the currently active folder for the target computer file system.

```
tg = slrt;
filesys_object = SimulinkRealTime.fileSystem(tg);
pwd(filesys_object)
```

```
ans =

  1×3 char array
```

```
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`

Output Arguments

`active_folder` — Currently active folder on the target computer

character vector

The path to the currently active folder on the target computer.

See Also

`SimulinkRealTime.fileSystem` | `cd` | `slrt`

Introduced in R2014a

removefile

Remove file from target computer

Syntax

```
removefile(filesys_object,file_name)
```

Description

`removefile(filesys_object,file_name)` removes a file from the target computer file system.

You cannot recover this file once you remove it.

Note: The `SimulinkRealTime.fileSystem` object will be removed in a future release. See the release note for file system commands to use instead. These commands use the `SimulinkRealTime.openFTP` function and the functions for the MATLAB `ftp` object.

Examples

Remove a File from the Target Computer

Remove `data2.dat` from the file system.

```
tg = slrt;
filesys_object = SimulinkRealTime.fileSystem(tg);
removefile(filesys_object,'data2.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`

`file_name` — Name of the file that is being removed

character vector | string scalar

The name of the file can be a name relative to the current folder or a fully qualified path. Enclose the file name in single quotation marks.

Example: `'data.dat'`

See Also

`SimulinkRealTime.fileSystem` | `fopen` | `slrt`

Introduced in R2014a

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.

Note: The `SimulinkRealTime.fileSystem` object will be removed in a future release. See the release note for file system commands to use instead. These commands use the `SimulinkRealTime.openFTP` function and the functions for the MATLAB `ftp` object.

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 `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 `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 `fclose`. If `C:\new_temp` does not exist, create it by using `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 | string scalar

The old name of the file or folder can be a name relative to the current folder or a fully qualified path.

Example: `'old_data.dat'`, `'C:\old_temp\old_data.dat'`

Data Types: `char` | `string`

`new_name` — New name of file or folder

character vector | string scalar

The new name of the file or folder can be a name relative to the current folder or a fully qualified path. 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` | `string`

See Also

`SimulinkRealTime.fileSystem` | `slrt`

Introduced in R2016a

rmdir

Remove empty folder from target computer

Syntax

```
rmdir(filesys_object,folder_name)
```

Description

`rmdir(filesys_object,folder_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.

You cannot recover this folder once you remove it.

Note: The `SimulinkRealTime.fileSystem` object will be removed in a future release. See the release note for file system commands to use instead. These commands use the `SimulinkRealTime.openFTP` function and the functions for the MATLAB `ftp` object.

Examples

Remove a Folder

Remove the folder `data2.dat` from the target computer file system.

```
tg = slrt;
filesys_object = SimulinkRealTime.fileSystem(tg);
rmdir(filesys_object,'data2.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`

`folder_name` — Name of a folder on the target computer
character vector | string scalar

If you omit the drive letter, the command assumes that the folder path is relative to the default drive.

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

Example: `logs`

Data Types: `char`

See Also

`SimulinkRealTime.fileSystem` | `mkdir` | `slrt`

Introduced in R2014a

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 or string scalar. If a drive with that name does not exist in the target computer, the function displays an error message.

Note: The `SimulinkRealTime.fileSystem` object will be removed in a future release. See the release note for file system commands to use instead. These commands use the `SimulinkRealTime.openFTP` function and the functions for the MATLAB `ftp` object.

Examples

Select the `C:\` Drive

Select the `C:\` drive in the target computer.

```
tg = slrt;
filesys_object = SimulinkRealTime.fileSystem(tg);
selectdrive(filesys_object,'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 on the target computer.

Example: `'C:\'`

Data Types: `char`

See Also

`SimulinkRealTime.fileSystem` | `slrt`

Introduced in R2016a

SimulinkRealTime.openFTP

Provide FTP access to folders and files on target computer

Description

The `SimulinkRealTime.openFTP` object provides FTP access to folders and files on the target computer.

Creation

`ftp_object = SimulinkRealTime.openFTP()` constructs and returns the FTP object corresponding to the default target computer. If you have only one target computer or if you designate a target computer as the default target computer in your system, use this form.

`ftp_object` `= SimulinkRealTime.fileSystem(`*`target_object`*`)` constructs and returns the file system object corresponding to the target computer that is accessible by *`target_object`*.

`ftp_object = SimulinkRealTime.fileSystem(`*`target_name`*`)` constructs and returns the file system object corresponding to the target computer that is accessible by *`target_name`*.

The `SimulinkRealTime.openFTP` function accepts these arguments:

- *`target_object`* — Object representing target computer.

  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.
- *`target_name`* — Name assigned to target computer.
- *`ftp_object`* — Object representing the target computer for FTP access.

  FTP object created by using the `SimulinkRealTime.openFTP` creation function.

The FTP object supports FTP access to the target computer file system. You work with the target computer file system from the development computer by using FTP methods.

Properties

No properties.

Examples

List Current Folder Contents on Default Target Computer

Create an ftp object for the default target computer, select an FTP folder, and list the contents of the current folder

```
TargetPC1Drive = SimulinkRealTime.openFTP();
cd(TargetPC1Drive, 'C:\');
dir(TargetPC1Drive)
```

```
KERNEL.SYS      E100BODI.COM    ODIPKT30.COM    USBVIEW.EXE     BOUNCIN1.DLM    logs
COMMAND.COM     E1000ODI.COM    PASSWORD.TEL    $RECYCLE.BIN    new_data.dat    rundir0
FDOS            FTPBIN.EXE      RTTBOOT.COM     dos.sg          nwr_tmp         loggingdb.json
AUTOEXEC.BAT    IPXODI.COM      RUNFTP.BAT      xpcboot.com     data.dat        sc3Integ.dat
FDCONFIG.SYS    LISTDEVS.EXE    SERDRV.SYS      xpmtgo.rtb      data1.dat       fLogData.dat
CONFIG.TEL      LPT1USB.SYS     TELPASS.EXE     XPCKRNL.RTB     slrtst.dat
DEVLOAD.COM     LSL.COM         TERMIN.COM      xPCTrace.csv    sc1Integ.dat
DOSUSB.COM      NET.CFG         USBDISK.SYS     sc2Integ.dat    private
```

Create FTP Object for Target Computer by Object

Create an FTP object for target computer by object `tg`, select the `C:\` drive on the target computer, and return current folder information.

```
tg = SimulinkRealTime.target('TargetPC1');
TargetPC1Drive = SimulinkRealTime.openFTP(tg);
cd(TargetPC1Drive, 'C:\');
cd(TargetPC1Drive)
```

```
ans =

    'C:\'
```

Create FTP Object for Target Computer by Name

Create an FTP object for target computer by name `TargetPC2`, select the `C:\` drive on the target computer, and return current folder information.

```
TargetPC2Drive = SimulinkRealTime.openFTP('TargetPC2');
cd(TargetPC2Drive, 'C:\');
cd(TargetPC2Drive)
```

```
ans =

    'C:\'
```

See Also

`ascii` | `binary` | `cd` | `close` | `delete` | `dir` | `ftp` | `mget` | `mkdir` | `mput` | `rename` | `rmdir`

Introduced in R2018a

SimulinkRealTime.SignalList

Create and manage signal list objects

Syntax

```
signalListObjectOut = SimulinkRealTime.SignalList()
add(signalListObjectIn,signalSelector)
remove(signalListObjectIn,signalSelector)
signalListObjectOut = validate(signalListObjectIn,rtApplication)
view(signalListObjectIn)
```

Description

`signalListObjectOut = SimulinkRealTime.SignalList()` creates a signal list object.

`add(signalListObjectIn,signalSelector)` adds a signal to the signal list object, by either a signal name or by a block path with a port index.

`remove(signalListObjectIn,signalSelector)` removes a signal from the signal list object, by either a signal name or by a block path with a port index.

`signalListObjectOut = validate(signalListObjectIn,rtApplication)` validates a signal list object.

`view(signalListObjectIn)` displays the list of signals in a signal list object.

Examples

Create a Signal List Object

Create a signal list object, `mySignals`.

```
mySignals=SimulinkRealTime.SignalList()
```

```
mySignals =

  SignalList with no properties.
```

Add a Signal by Name to Signal List Object

For model `xpcosc`, create a signal list object `mySignals`, name the signal `Integ_out`, and add the signal by name to the signal list.

```
mySignals=SimulinkRealTime.SignalList();
p = get_param('xpcosc/Integrator','PortHandles');
l = get_param(p.Outport,'Line');
set_param(l,'Name','Integ_out');
add(mySignals,'Integ_out');
```

Add a Signal by Block Path with Port Index to Signal List Object

For model `xpcosc`, create signal list object `mySignals` and add the signal by a block path with a port index to the signal list.

```
mySignals=SimulinkRealTime.SignalList();
add(mySignals,'xpcosc/Integrator',1);
```

View Signals in Signal List Object

For model `xpcosc` with named signals `Integ_out` and `Integ1_out`, list the signals in the signal list object `mySignals`.

```
view(mySignals)

Integ_out
Integ1_out
```

Validate Signal List Object

For input signal list object `mySignals` that contains named signals `Integ_out`, `Integ1_out`, and `Integ2_out`, check whether the named signals are available in real-

time application `xpcosc`. Any unavailable signals are added to the output signal list object `unavailSignals`.

```
unavailSignals=validate(mySignals,'xpcosc');
view(unavailSignals)
```

```
Integ2_out
```

Remove a Signal from a Signal List Object

For input signal list object `mySignals` that contains named signal `Integ2_out`, remove the signal from the signal list object.

```
remove(mySignals,'Integ2_out')
```

Input Arguments

`signalListObjectIn` — Select the signal list object to modify

`signalListObject`

The *`signalListObject`* argument selects the signal list object that the function modifies by adding or by removing signals.

Example: `mySignals`

`signalSelector` — Select the signal to add or remove

`signalName` | *`blockPath,portIndex`*

The *`signalSelector`* argument identifies the signal that the function adds or removes from the signal list object. You can specify the selector by *`signalName`* or by the combination of *`blockPath,portIndex`*.

Example: `'Integ_out'`

Example: `'xpcosc/Integrator'`

`rtApplication` — Select the real-time application for signal list

`rtApplicationName`

The *rtApplicationName* argument identifies the real-time application that contains the signals listed in the input signal list object. The validation identifies any signals in the input signal list that are not available in the real-time application.

Example: `xpcosc`

Output Arguments

signalListObjectOut — Select the signal list object to create
signalListObject

The *signalListObjectOut* argument selects the signal list object that is created by the `SimulinkRealTime.SignalList()` function or by the `validate()` function.

See Also

`setStreamingSignals`

Topics

“Stream Signal Data from Target Computer to Simulation Data Inspector”

Introduced in R2019a

setStreamingSignals

Select target computer and list of signals for streaming data to the Simulation Data Inspector (SDI)

Syntax

```
setStreamingSignals(target, signalListObject)
setStreamingSignals(target, [])
```

Description

`setStreamingSignals(target, signalListObject)` selects a target computer and starts streaming signal data to the Simulation Data Inspector from the signals in the signal list object.

`setStreamingSignals(target, [])` selects a target computer and stops streaming signal data to the Simulation Data Inspector by identifying an empty array of signal list objects.

Examples

Start Streaming Signal Data

Selects target computer `tg` and start streaming signal data to the Simulation Data Inspector from the signals in signal list object `mySignals`.

```
setStreamingSignals(tg,mySignals)
```

Stop Streaming Signal Data

Select target computer `tg` and stop streaming signal data to the Simulation Data Inspector by selecting an empty array of signal list objects.

```
setStreamingSignals(tg,[])
```

Input Arguments

`target` — Selects the target computer

target computer name

The *`target`* argument selects the target computer that provides the streaming signal data from the real-time application.

Example: `tg`

`signalListObject` — Selects the signal list object

signal list object

The *`signalListObject`* argument selects the signal list that provides the streaming signal data to the Simulation Data Inspector.

Example: `mySignals`

See Also

`SimulinkRealTime.SignalList`

Topics

“Stream Signal Data from Target Computer to Simulation Data Inspector”

Introduced in R2019a

SimulinkRealTime.target

Represent real-time application and target computer status

Description

A `SimulinkRealTime.target` object represents a 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.
- Start, stop, and retrieve information from the profiler.

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.

Creation

`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, check 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`.

The `SimulinkRealTime.target` function accepts these arguments:

- `target_name` — Name assigned to target computer (character vector or string scalar). For example, `'TargetPC1'`.
- `target_object` — Object representing target computer. For example, `tg`.

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.

This example creates a target object that communicates with the default target computer and reports 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
```

This example creates a target object that communicates with target computer `TargetPC1` and reports the status of the target computer. In this case, the target computer is not connected to the development computer.

```
target_object = SimulinkRealTime.target('TargetPC1')
```

Real-Time Application Properties

Target Computer

`Application` — Name of real-time application

`'loader'` | character vector | string scalar

This property is read-only.

Name of real-time application running on target computer, specified as a character vector or string scalar. This name 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'` | character vector | string scalar

This property is read-only.

Communication status between the development and target computers, specified as character vector or string scalar.

`CPUoverload` — CPU status for overload
`'none'` (default) | `'detected'` | character vector | string scalar

This property is read-only.

CPU status for overload, specified as a character vector or string scalar. If the real-time application requires more CPU time than the model sample time provides, the kernel changes this value from `'none'` to `'detected'`. It then stops the current run. To keep this status from changing to `'detected'`, you must user a faster processor or specify a larger sample time.

`Mode` — Execution mode of the real-time application
`'Real-Time Singletasking'` (default) | `'Real-Time Multitasking'` | character vector | string scalar

This property is read-only.

Execution mode of the real-time application on the target computer, specified as a character vector or string scalar. 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.

`WorkingDirectory` — Current working folder of the target computer

character vector | string scalar

Current working folder on the target computer, specified as a character vector (for example, `'C:\'`) or string scalar.

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'` | character vector | string scalar

This property is read-only.

Execution status of real-time application, specified as character vector or string scalar.

`StopTime` — Time when real-time application stops running

seconds | `'Inf'`

Time when the real-time application stops running, specified in seconds, as character vector, or as string scalar. 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 of double

This property is read-only.

Storage in the MATLAB workspace for task execution time, specified as a vector of double.

Signal Visualization

`LogMode` — Controls which data points are logged

`'Normal'` (default) | double

The values are the following meaning:

- `'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'` | character vector | string scalar

Flag set to view the list of signals from your Simulink model, specified as character vector or string scalar. 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, displayed 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'` | character vector | string scalar

Flag set to view the list of parameters from your Simulink model, specified as character vector or string scalar. MATLAB displays the parameter list when you display the properties for a target object.

Profiler

`ProfilerStatus` — State of profiler

`Ready` (default) | `Running` | `DataAvailable`

The profiler states have the following meaning:

- `Ready` — The profiler starts in this state and remains in it until the profiler runs.

  If the profiler runs, it reenters this state if the profiler:

  - Stopped without collecting data.
  - Collected data and the data was downloaded to the development computer.
  - Was reset.
- `Running` — The command to start the profiler succeeded.

  If a real-time application is running, the profiler collects data.

  If a real-time application is not running, the profiler initializes and waits. When a real-time application starts, the profiler starts collecting data.
- `DataAvailable` — The command to stop the profiler succeeded. The profiler collected data, but the data has not been downloaded to the development computer. In this state, an attempt to restart the profiler produces the following results:

  - If an application is running, calling this function returns an error. Download the data or reset the profiler before restarting it.
  - If an application is not running, calling this function restarts the profiler, and this operation discards the existing profile data from the target computer.

Object Functions

ping	Test communication between development and target computers
reboot	Restart target computer
close	Close connection between development and target computers
createStandAlone	Create a standalone real-time application for target computer
load	Download real-time application to target computer
unload	Remove real-time application from target computer
start	Starts execution of real-time application on target computer
stop	Stops execution of real-time application on target computer
addscope	Create a scope of specified type
getscope	Return scope identified by scope number
remscope	Remove scope from target computer
getlog	Portion of output logs from target object

importLogData	Import buffered logging data to the active session of the Simulation Data Inspector
getsignal	Value of signal
getsignalid	Signal index from signal hierarchical name
getsignalidsfromlabel	Vector of signal indices
getsignallabel	Signal label for signal index
getsignalname	Signal name from index list
getparam	Read value of observable parameter in real-time application
setparam	Change value of tunable parameter in real-time application
getparamid	Parameter index from parameter hierarchical name
getparamname	Block path and parameter name from parameter index
loadparamset	Restore parameter values saved in specified file
saveparamset	Save real-time application parameter values
startProfiler	Start profiling service on target computer
stopProfiler	Stop profiling service on target computer
getProfilerData	Retrieve profile data object
resetProfiler	Reset profiling service state to Ready
getDiskSpace	Return free space and total space on the drive, in bytes

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 = slrt('TargetPC1');
load(tg,ex_model);
tg
```

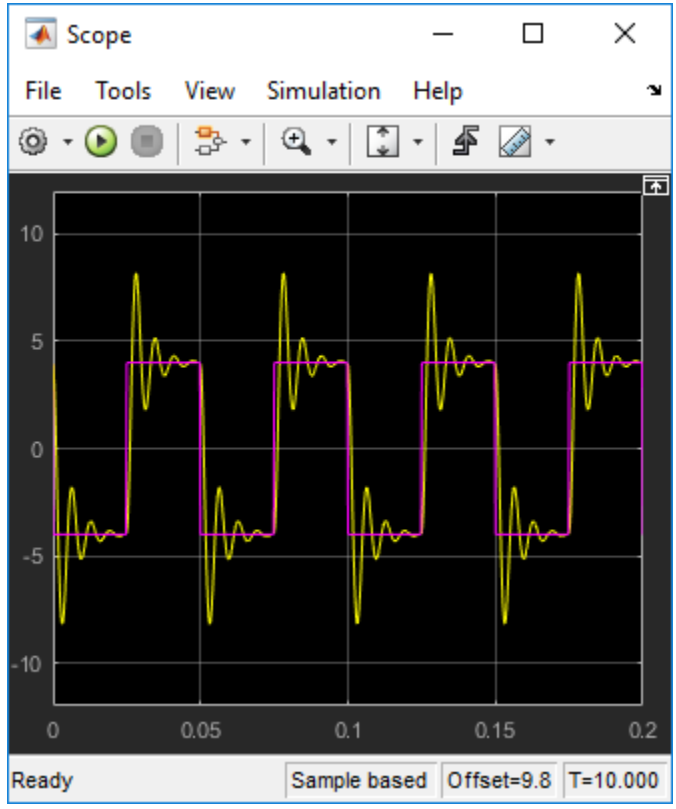
```
Target: TargetPC1
   Connected            = Yes
   Application          = xpcosc
   Mode                 = Real-Time Single-Tasking
```

```
Status               = stopped
CPUOverload          = none

ExecTime             = 0.0000
SessionTime          = 769.0726
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
ProfilerStatus       = Ready

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 figure.



Unload real-time application

```
unload(tg)
```

```
Target: TargetPC1
   Connected            = Yes
   Application          = loader
```

See Also

“Target Computer Commands” | `Profiler Data`

Topics

“Execution Profiling for Real-Time Applications”

“Blocks Whose Outputs Depend on Inherited Sample Time” (Simulink)

Introduced in R2014a

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

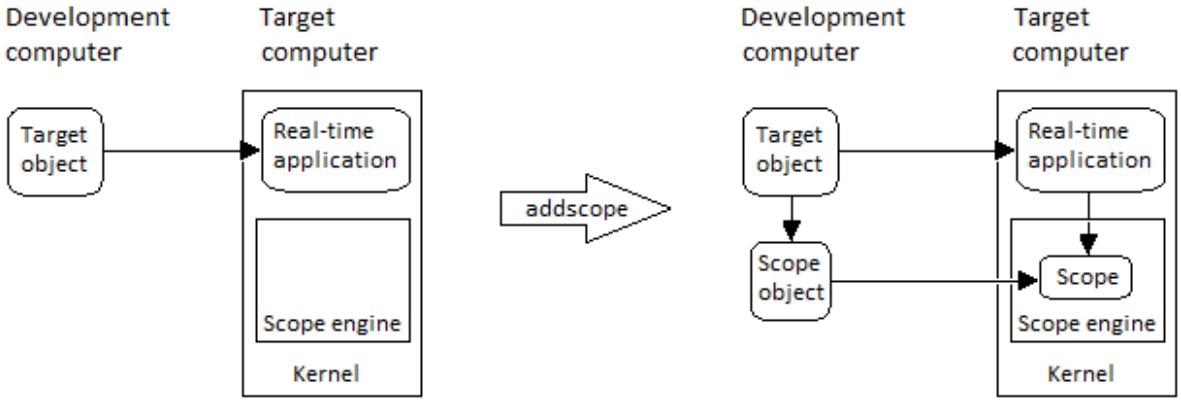
`scope_object = addscope(target_object)` creates on the target computer a host scope and assigns as its scope number the next available integer in the target object property `Scopes`. It returns the object representing this scope.

`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

Create two target scopes 3 and 4 using a vector of scope numbers and assign the scope objects to variable `scope_object_vector`.

```
tg = slrt;
scope_object_vector = addscope(tg, 'target', [3, 4])

scope_object_vector =

Simulink Real-Time Scope
   Application          = xpco   ScopeId             = 3
   Status               = Interrupted
   Type                 = Target
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = -1
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 3
   TriggerSample        = 0
   DisplayMode          = Redraw (Graphical)
   YLimit               = Auto
   Grid                 = on
   Signals              = no Signals defined

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 4
   Status               = Interrupted
   Type                 = Target
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = -1
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 4
   TriggerSample        = 0
   DisplayMode          = Redraw (Graphical)
   YLimit               = Auto
```

```
Grid                = on
Signals             = no Signals defined
```

Input Arguments

`target_object` — Object representing target computer

`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`

`scope_type` — Type of scope to create

`'host'` (default) | `'target'` | `'file'`

Type of scope to create, as a character vector or string scalar. 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” | `SimulinkRealTime.target` | `getscope` | `remscope`

Introduced in R2014a

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`

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

`SimulinkRealTime.target` | `reboot`

Introduced in R2014a

createStandAlone

Create a standalone real-time application for target computer

Syntax

```
createStandAlone(target_object,modelName)
```

Description

`createStandAlone(target_object,modelName)` creates a standalone real-time application. Before using this method, you must build the model and create a target object.

For more information about standalone real-time applications, see “Standalone Mode”.

Examples

Create StandAlone Real-Time Application

Build the model, create a target object, and create a standalone real-time application.

```
rtwbuild('model_name');
tg = slrt('target_name');
createStandAlone(tg,'model_name');
```

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`

`modelName` — Name of model
character vector

Name of the model from which the real-time application was built.

Example: `'my_Model'`

See Also

"Target Computer Commands" | `SimulinkRealTime.target`

Topics

"Standalone Mode"

Introduced in R2019b

getDiskSpace

Return free space and total space on the drive, in bytes

Syntax

```
disk_space = getDiskSpace(target_object, drive_name)
```

Description

`disk_space = getDiskSpace(target_object, drive_name)` returns a structure containing the free space and the total space on the drive, in bytes. If a drive with that name does not exist on the target computer, the function displays an error message.

Examples

Display the Disk Space for the `C:\` Drive

Return the free space and the total space on the `C:\` drive on the target computer.

```
tg = slrt;
disk_space = getDiskSpace(tg,'C:\')
```

```
disk_space =

  struct with fields:

     freeDiskSpacebytes: 5.9885e+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`

`drive_name` — Name of the drive to access
character vector | string scalar

Enclose the drive name as a character vector or string scalar. The drive must exist on the target computer.

Example: `'C:\'`

Data Types: `char` | `string`

Output Arguments

`disk_space` — Returns a structure containing fields with free space and total space on the drive
struct

See Also

`SimulinkRealTime.openFTP` | `SimulinkRealTime.target` | `slrt`

Introduced in R2018a

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 a decimation of `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 Decimated Values Starting from Offset

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 a Value

Read one 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`

`log_name` — Defines 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 outport blocks
- `TETLog` — Task execution times (TET)

Example: `'Timelog'`

Data Types: `char`

`first_point` — Sample from which to start retrieving data
1 (default) | positive integer

If specified without `number_samples`, this parameter returns only the value at `first_point`.

Example: 10

`number_samples` — Number of samples to retrieve
all points in log (default) | positive integer

Number of samples to retrieve starting with `first_point`, after decimation.

Example: `10`

`decimation` — Retrieve every `decimation`th value
1 (default) | positive integer

If `1`, returns all sample points. If `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

See Also

`SimulinkRealTime.target` | `importLogData`

Topics

“Set Configuration Parameters”

Introduced in R2014a

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 global parameter `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'`.

```
% If you previously built a real-time application from model
% slrttestmdl, you can load the application on the target
% computer and get parameter information from the application.
%
% When working with parameters in the real-time application, it
```

```
% can be helpful to display the parameters that are available
% in the application by setting the ShowParameters property to
% 'on' with the command:
% tg.ShowParameters = 'on'
%
tg = slrt;
load(tg, 'slrttestmdl');
getparam(tg, 'Signal Generator', 'Amplitude')

ans =

     4
```

Get Global Parameter by Scalar Parameter Name

Get the value of MATLAB variable `'Freq'`.

```
% To create a variable and assign the Frequency parameter
% to it, use the block parameter dialog box or commands such as:
% Freq = 20;
% set_param('slrttestmdl/Signal Generator', 'Frequency', 'Freq');
%
% If you previously created a variable Freq, assigned the Frequency
% parameter value to Freq, and built a real-time application from model
% slrttestmdl, you can load the application on the target
% computer and get parameter information from the application.
%
tg = slrt;
load(tg, 'slrttestmdl')
getparam(tg, 'Freq')

ans =

     20
```

Get Global Parameter by Parameter Structure Name

Get the value of parameter structure `'oscp'`.

```
% To create an array of gain values and assign the gain
% parameters to its elements, use the block parameter dialog
```

```
% box or commands such as:
% oscp.G0 = 1000000;
% oscp.G1 = 400;
% oscp.G2 = 1000000;
% set_param('slrttestmdl/Gain', 'Gain', 'oscp.G0');
% set_param('slrttestmdl/Gain1', 'Gain', 'oscp.G1');
% set_param('slrttestmdl/Gain2', 'Gain', 'oscp.G2');
%
% If you previously created a structure array of gain values,
% assigned the the gain parameter values to the array elements,
% and built a real-time application from model slrttestmdl, you
% can load the application on the target computer and get parameter
% information from the application.
%
tg = slrt;
load(tg, 'slrttestmdl')
getparam(tg, 'oscp')

ans =

    G0: 1000000
    G1: 400
    G2: 1000000
```

Get Global Parameter by Parameter Structure Field Name

Get the value of MATLAB variable `'oscp.G2'`.

```
% To create a structure array of gain values and assign the gain
% parameters to its elements, use the block parameter dialog box
% or commands such as:
% oscp.G2 = 1000000;
% set_param('slrttestmdl/Gain2', 'Gain', 'oscp.G2');
%
% If you previously created a structure array of gain values,
% assigned the the gain parameter values to the array elements,
% and built a real-time application from model slrttestmdl, you
% can load the application on the target computer and get parameter
% information from the application.
%
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.

```
% If you previously built a real-time application from model
% slrttestmdl, you can load the application on the target
% computer and get parameter information from the application.
%
tg = slrt;
pid = getparamid(tg, 'Gain', 'Gain');

getparam(tg, pid)
```

```
ans =

     1000000
```

Get Global Parameter by Parameter Index

Get the parameter index of MATLAB variable `'G2'`, and then get its value.

```
% To create a structure array of gain values and assign the
% gain parameters to its elements, use the block parameter
% dialog box or commands such as:
% G2 = 1000000;
% set_param('slrttestmdl/Gain2', 'Gain', 'G2');
%
% If you previously created a variable G2, assigned the
% the variable to the Gain2 block gain parameter, and built a
% real-time application from model slrttestmdl, you can load the
% application on the target computer and get parameter information
% from the application.
%
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`

`parameter_block_name` — Hierarchical name of the originating block

character vector | string scalar

The empty character vector (`''`) or empty string scalar (`""`) as a block name marks a global parameter 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 | string scalar

The parameter can designate either a block parameter or a global parameter that provides the value for a block parameter. To be accessible via parameter name, the block parameter or MATLAB variable 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 global parameter that provides the value for a block parameter. To be accessible via parameter index, the block parameter or MATLAB variable must be observable.

To access a parameter index, type `tg.ShowParameters = 'on'` in the Command Window, and count lines starting with `0`.

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.

See Also

`SimulinkRealTime.target` | `getparamid` | `setparam`

Topics

“Tunable Block Parameters and Tunable Global Parameters”
“Troubleshoot Parameters Not Accessible by Name”

Introduced in R2014a

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 global parameter that provides the value for a block parameter. The function searches the parameter list by the MATLAB variable name. The name is case-sensitive.

For the block name argument, enter the empty character vector (`''`) or empty string scalar (`""`).

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 Global Parameter 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`

`parameter_block_name` — Hierarchical name of the originating block
character vector | string scalar

The empty character vector (`''`) or empty string scalar (`""`) as a block name marks a global parameter 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 | string scalar

The parameter can designate either a block parameter or a global parameter that provides the value for a block parameter. To be accessible via parameter name, the block parameter or MATLAB variable must be observable.

Note Simulink Real-Time does not support parameters of multiword data types.

Example: `'Gain'`, `'oscp.G1'`, `'oscp'`, `'G2'`

Output Arguments

`parameter_index` — Index number of the parameter
nonnegative integer

The parameter index can mark either a block parameter or a global parameter that provides the value for a block parameter. To be accessible via parameter index, the block parameter or MATLAB variable must be observable.

To access a parameter index, type `tg.ShowParameters = 'on'` in the Command Window, and count lines starting with `0`.

Note Parameter access by parameter index will be removed in a future release. Access parameters by parameter name instead.

Example: `0`, `1`

See Also

`SimulinkRealTime.target` | `getparam` | `setparam`

Topics

“Tunable Block Parameters and Tunable Global Parameters”
“Troubleshoot Parameters Not Accessible by Name”

Introduced in R2014a

getparamname

Block path and parameter name from parameter index

Syntax

```
[block_path, parameter_name] = getparamname(target_object,
parameter_index)
```

Description

`[block_path, parameter_name] = getparamname(target_object, parameter_index)` returns a vector containing the block path and the parameter name for the parameter specified by `parameter_index`.

Examples

Get Block Path and Parameter Name for Parameter

Extract the block path and parameter name for parameter 6 of `'ex_slrt_sf_car'`.

```
tg = slrt;
[block_path, parameter_name] = getparamname(tg,6)
```

```
block_path =

  1×17 char array

Engine/Integrator


parameter_name =

  1×16 char array
```

`InitialCondition`

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`

`parameter_index` — Index number of the parameter

nonnegative integer

The parameter index can mark either a block parameter or a global parameter that provides the value for a block parameter. To be accessible via parameter index, the block parameter or MATLAB variable must be observable.

To access a parameter index, type `tg.ShowParameters = 'on'` in the Command Window, and count lines starting with `0`.

Note Parameter access by parameter index will be removed in a future release. Access parameters by parameter name instead.

Example: `0`, `1`

Output Arguments

`block_path` — Hierarchical path to block containing parameter

character vector

The path consists of nested blocks separated by `'/'`.

`parameter_name` — Name of parameter in block

character vector

The parameter name as given in the block properties list.

See Also

`SimulinkRealTime.target`

Introduced in R2014a

getProfilerData

Retrieve profile data object

Syntax

```
profiler_object = getProfilerData(target_object)
profiler_object = getProfilerData(target_object);
```

Description

`profiler_object = getProfilerData(target_object)` downloads the profiler files from the target computer to the development computer and assigns the data to `profiler_object`. It displays an execution profile plot and a code execution profiling report.

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 lists the model sections. The numbers underneath the bars indicate the processor cores.

The Code Execution Profiling Report displays model execution profile results for each task.

- To display the profile data for a section of the model, click the **Membrane** button next to the section.
- To display the TET data for the section in the Simulation Data Inspector, click the **Plot time series data** button .
- To view the section in Simulink Editor, click the link next to the **Expand Tree** button .
- To view the lines of generated code corresponding to the section, click the **Expand Tree** button and then click the **View Source** button .

`profiler_object = getProfilerData(target_object);` downloads the profiler files from the target computer to the development computer and assigns the data to

`profiler_object`. To display the profiler results, call the `plot` and `report` functions with the `profiler_object` as argument.

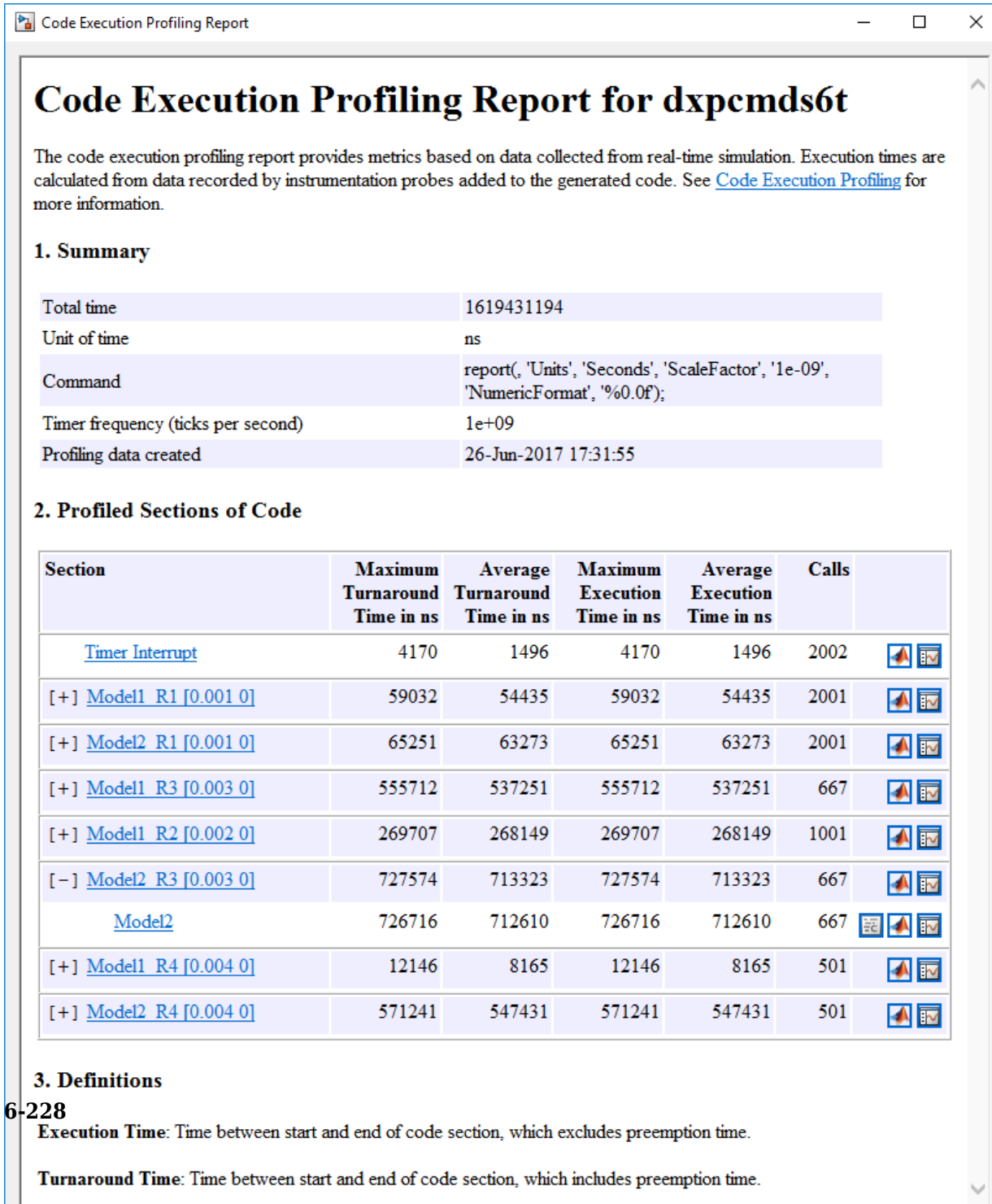
Examples

Run Profiler and Implicitly Display Profiler Data

Starts the profiler, stops the profiler, and displays results data. The real-time application `dxpcmds6t` is already loaded.

```
tg = slrt;
startProfiler(tg);
start(tg);

stopProfiler(tg);
stop(tg);

profiler_object = getProfilerData(tg)
```

```
Processing data, please wait ...
Code execution profiling data for model dxpcmds6t.
```



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	1619431194
Unit of time	ns
Command	report(, 'Units', 'Seconds', 'ScaleFactor', '1e-09', 'NumericFormat', '%0.0f');
Timer frequency (ticks per second)	1e+09
Profiling data created	26-Jun-2017 17:31:55

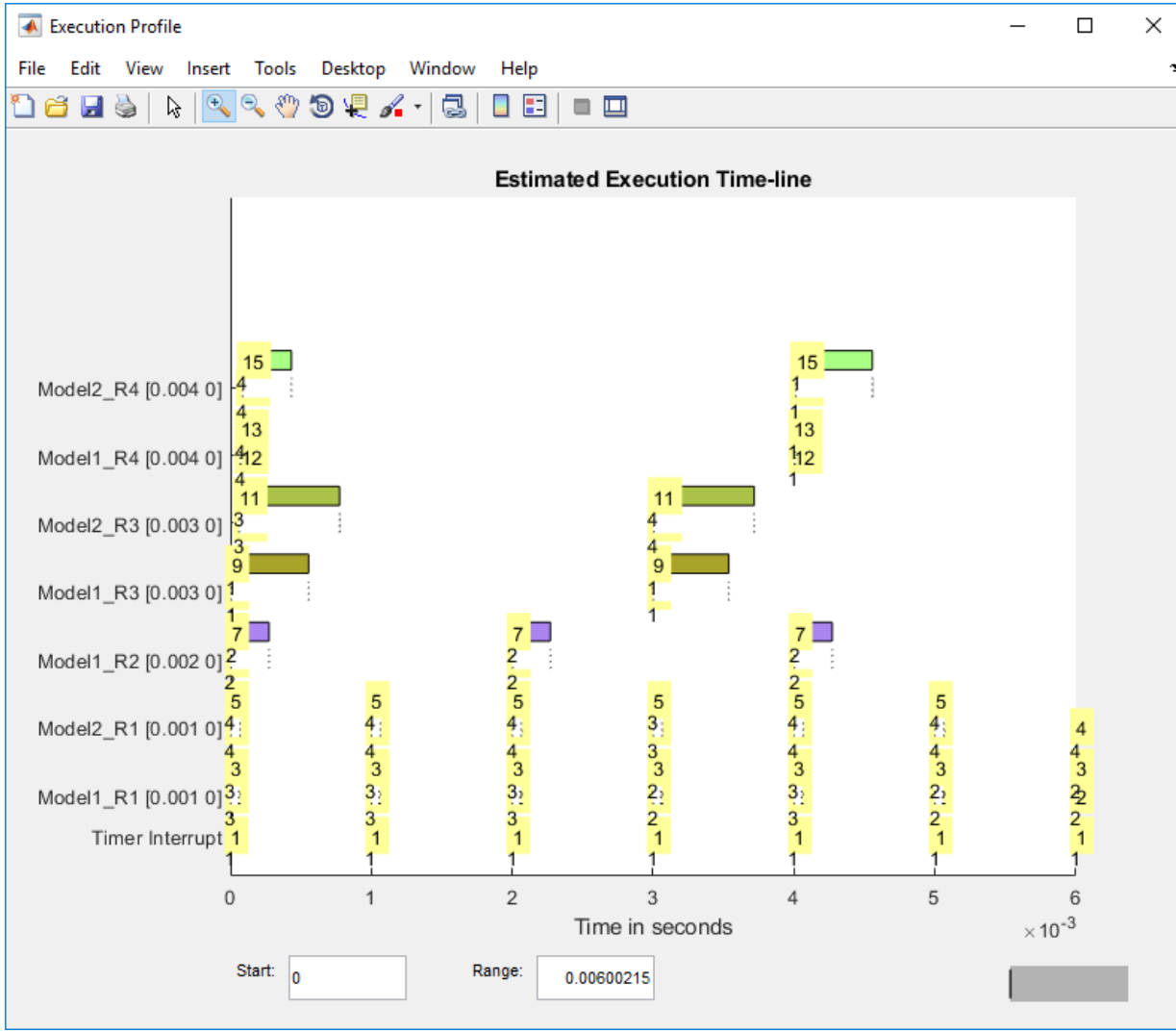
2. Profiled Sections of Code

Section	Maximum Turnaround Time in ns	Average Turnaround Time in ns	Maximum Execution Time in ns	Average Execution Time in ns	Calls
Timer Interrupt	4170	1496	4170	1496	2002
[+] Model1_R1 [0.001 0]	59032	54435	59032	54435	2001
[+] Model2_R1 [0.001 0]	65251	63273	65251	63273	2001
[+] Model1_R3 [0.003 0]	555712	537251	555712	537251	667
[+] Model1_R2 [0.002 0]	269707	268149	269707	268149	1001
[−] Model2_R3 [0.003 0]	727574	713323	727574	713323	667
Model2	726716	712610	726716	712610	667
[+] Model1_R4 [0.004 0]	12146	8165	12146	8165	501
[+] Model2_R4 [0.004 0]	571241	547431	571241	547431	501

3. Definitions

Execution Time: Time between start and end of code section, which excludes preemption time.

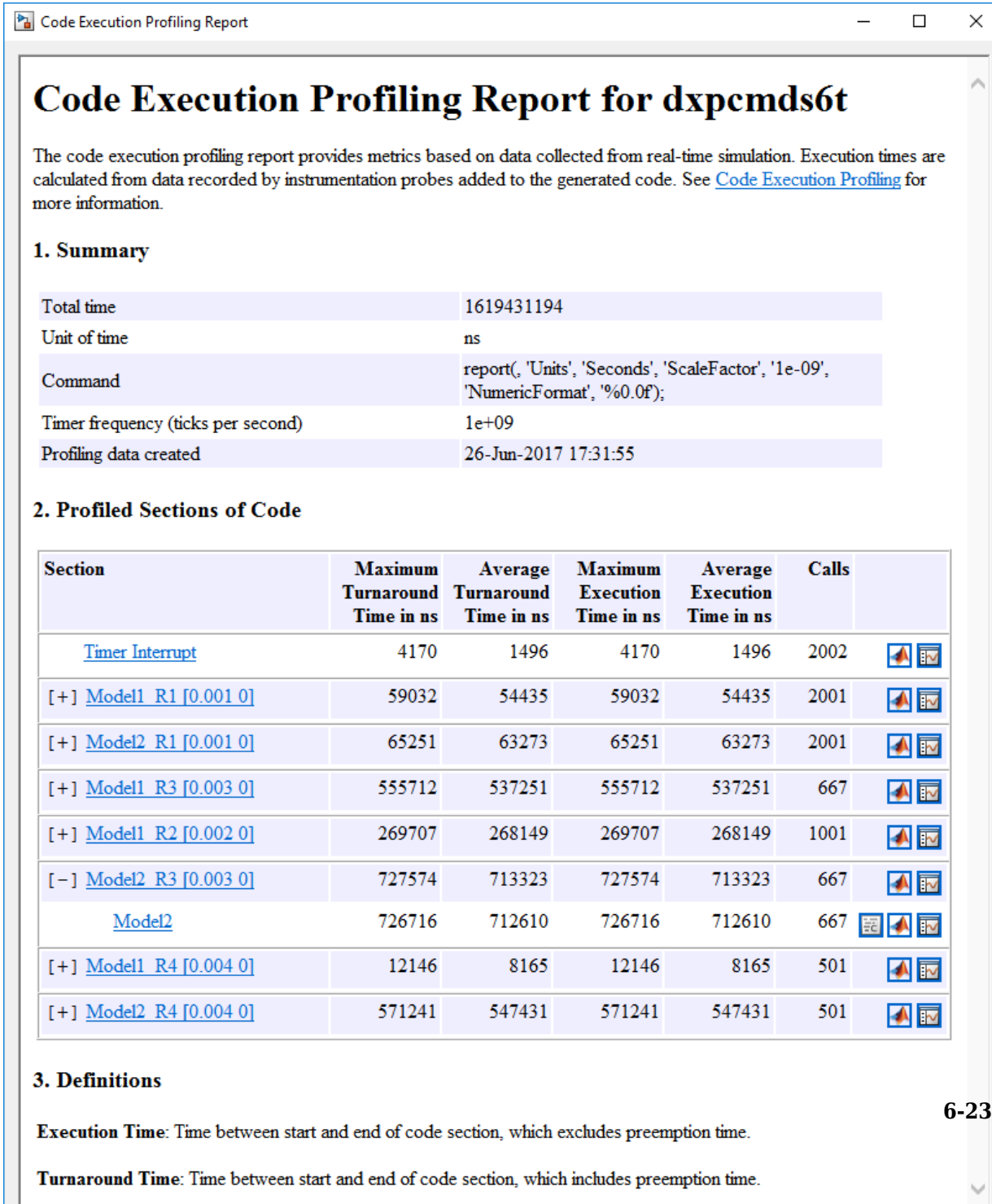
Turnaround Time: Time between start and end of code section, which includes preemption time.



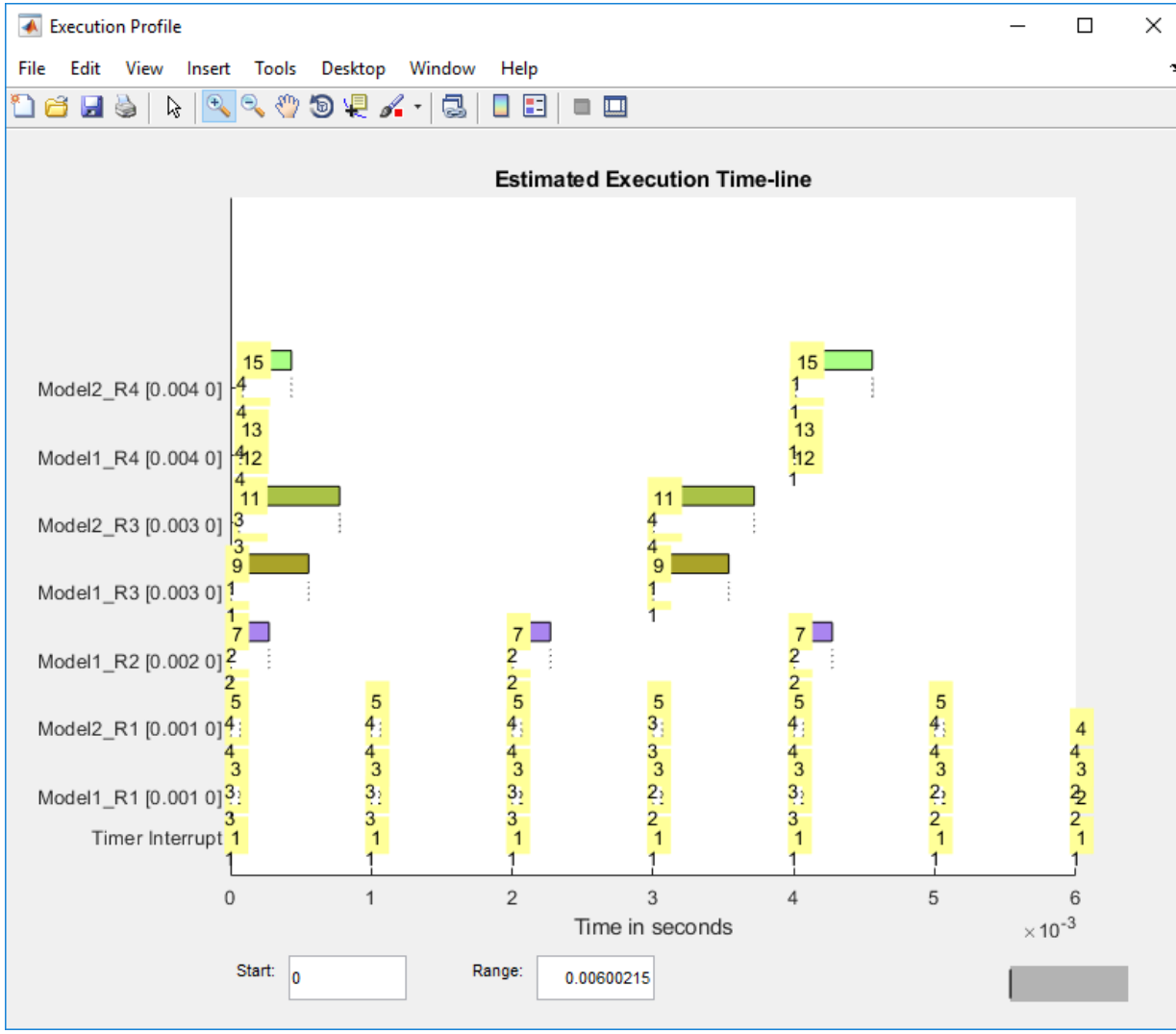
Run Profiler and Explicitly Display Profiler Data

Starts the profiler, stops the profiler, and retrieves results data. Calls `report` and `plot` on the results data. The real-time application `dxpcmds6t` is already loaded.

```
tg = slrt;
startProfiler(tg);
start(tg);

stopProfiler(tg);
stop(tg);

profiler_object = getProfilerData(tg);

Processing data, please wait ...

report(profiler_object);
```


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 1619431194
Unit of time ns
Command report(, 'Units', 'Seconds', 'ScaleFactor', '1e-09', 'NumericFormat', '%0.0f');
Timer frequency (ticks per second) 1e+09
Profiling data created 26-Jun-2017 17:31:55
2. Profiled Sections of Code
Section | Maximum Turnaround Time in ns | Average Turnaround Time in ns | Maximum Execution Time in ns | Average Execution Time in ns | Calls
Timer Interrupt 4170 1496 4170 1496 2002
[+] Model1_R1 [0.001 0] 59032 54435 59032 54435 2001
[+] Model2_R1 [0.001 0] 65251 63273 65251 63273 2001
[+] Model1_R3 [0.003 0] 555712 537251 555712 537251 667
[+] Model1_R2 [0.002 0] 269707 268149 269707 268149 1001
[−] Model2_R3 [0.003 0] 727574 713323 727574 713323 667
Model2 726716 712610 726716 712610 667
[+] Model1_R4 [0.004 0] 12146 8165 12146 8165 501
[+] Model2_R4 [0.004 0] 571241 547431 571241 547431 501
3. Definitions
Execution Time: Time between start and end of code section, which excludes preemption time.
Turnaround Time: Time between start and end of code section, which includes preemption time.

```
plot(profiler_object);
```



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`

Output Arguments

`profiler_object` — Contains the profiler result
structure

MATLAB variable using which you can access the result of the profiler execution. You display the profiler data by calling the `plot` and `report` functions.

The structure has the following fields:

- `TargetName` — Name of target computer in target computer settings.
- `ModelInfo` — Information about model on which profiler ran:
  - `ModelName` — Name of real-time application.
  - `MATLABRelease` — MATLAB release under which model was built.
  - `KernelStamp` — Timestamp of target computer kernel build.
  - `Display` — Display mode of target computer kernel. One of `Graphics` and `Text`.
  - `BootMode` — Boot mode of target computer kernel. One of `Normal` and `Standalone`.

You can access the data in the *`profiler_object`* variable. To access the profiler data, before running the profiler, open the **Configuration Parameters**. In the **Real-Time** tab, click **Hardware Settings**. Select the **Code Generation > Verification > Workspace variable** option and set the value to `executionProfile`. Select the **Save options** option and set the value to `All data`. After running the profiler, use the technique described for the `Sections` function.

See Also

Enable Profiler | `SimulinkRealTime.ProfilerData` | `SimulinkRealTime.target` | `stopProfiler`

Topics

“Execution Profiling for Real-Time Applications”

Introduced in R2017b

importLogData

Import buffered logging data to the active session of the Simulation Data Inspector

Syntax

```
importLogData(target_object)
```

Description

`importLogData(target_object)` imports buffered logging data to the active session of the Simulation Data Inspector immediately. Without using this function, the Simulation Data Inspector imports the data when the stop time is reached for the run of the real-time application. A limitation is that a small gap in the logged data appears in the Simulation Data Inspector at the time that you use the `importLogData` function.

Examples

Import Buffered Log Data to the Simulation Data Inspector

To import buffered logging data into the active session of the Simulation Data Inspector immediately, use these steps.

1. Open the `xpcFileLogging` model.

   ```
   open_system(fullfile(matlabroot,'toolbox','rtw','targets','xpc','xpcdemos','xpcFile
   ```

2. Mark a signal for logging.
3. Set the logging properties to buffered
4. Set the model stop time to `'Inf'`

   ```
   set_param('xpcFileLogging','StopTime','Inf');
   ```

5. Build and download the real-time application
6. Create a target object for the default target computer with the command:

```
tg = slrt;
```

7 Run the real-time application.

```
start(tg);
```

8 Create a new Simulation Data Inspector run.

```
Simulink.sdi.Run.create
```

9 Import the buffered logged data into the Simulation Data Inspector

```
 importLogData(tg)
```

The status message indicates that the log data is being transferred.

```
Transferring logging data for model xpcFileLogging, please wait ...
```

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`

See Also

`SimulinkRealTime.target` | `getlog`

Introduced in R2018a

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 (PCI device number)
- PCI function 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, check 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 calling the function `SimulinkRealTime.pingTarget`.

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 Supported Devices on Default Computer

Start the default target computer with the Simulink Real-Time kernel. Check the connection between the development and the target computer. At the command prompt, type the command on the development computer. The command returns all supported devices other than Ethernet cards.

```
slrtpingtarget
```

```
target_object = slrt;
getPCIInfo(target_object)
```

```
List of installed PCI devices:

General Standards        PMC-ADADIO
     Bus 6, Slot 4, Function 0, IRQ 10
     AI AO DI DO
     VendorID 0x10b5, DeviceID 0x9080, SubVendorID 0x10b5, ...
         SubDeviceID 0x2370
     A/D Chan: 0, D/A Chan: 4, DIO Chan: 8
     Released in: R14SP2 or Earlier
     Notes: Uses Compact PCI and PCI carriers
.
.
.
```

Display Information for Ethernet Controllers on Default Computer

Start the default target computer with the Simulink Real-Time kernel. Check 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                    82579LM
     Bus 0, Slot 25, Function 0, IRQ 3
     Ethernet controller
     VendorID 0x8086, DeviceID 0x1502, SubVendorID 0x15bd, ...
```

```
         SubDeviceID 0x100a
     Released in: R2012b
     Notes: Intel 8254x Gigabit Ethernet series

Intel                    82574L
     Bus 5, Slot 0, Function 0, IRQ 10
     Ethernet controller
     VendorID 0x8086, DeviceID 0x10d3, SubVendorID 0x15bd, ...
         SubDeviceID 0x100a
     Released in: R2010a
     Notes: Intel 8254x Gigabit Ethernet series
```

Display Information for All Devices on Default Computer

Start the default target computer with the Simulink Real-Time kernel. Check 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, Function 0, IRQ 0
     Host Bridge
     VendorID 0x8086, DeviceID 0x0150, SubVendorID 0x8086, ...
         SubDeviceID 0x0150
.
.
.
Intel                    82579LM
     Bus 0, Slot 25, Function 0, IRQ 3
     Ethernet controller
     VendorID 0x8086, DeviceID 0x1502, SubVendorID 0x15bd, ...
         SubDeviceID 0x100a
     Released in: R2012b
     Notes: Intel 8254x Gigabit Ethernet series.
.
```

```
.
.
```

Display Verbose Information for All Devices on Default Computer

Start the default target computer with the Simulink Real-Time kernel. Check 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, Function 0, IRQ 0
     Host Bridge
     VendorID 0x8086, DeviceID 0x0150, SubVendorID 0x8086, ...
         SubDeviceID 0x0150
     BaseClass 6, SubClass 0

Intel                    Unknown
     Bus 0, Slot 1, Function 0, IRQ 10
     PCI-to-PCI Bridge
     VendorID 0x8086, DeviceID 0x0151, SubVendorID 0x0000, ...
         SubDeviceID 0x0000
     BaseClass 6, SubClass 4
     BAR BaseAddress AddressSpace       MemoryType PreFetchable
      2)       10100       Memory   32-bit decoder           no
      3)          F0       Memory   32-bit decoder           no
      4)        FFF0       Memory   32-bit decoder           no
      5)       1FFF0          I/O
.
.
.
```

Return Information for Supported Devices on Default Computer

Start the default target computer with the Simulink Real-Time kernel. Check the connection between the development and target computers. At the command prompt,

type the command on the development computer. The command returns all supported devices other than Ethernet cards. Display a structure in the vector.

```
slrtpingtarget
```

```
target_object = slrt;
pci_devices = getPCIInfo(target_object);
pci_devices(16)
```

```
ans =

  struct with fields:

             Bus: 6
            Slot: 4
        Function: 0
        VendorID: '10B5'
        DeviceID: '9080'
     SubVendorID: '10B5'
     SubDeviceID: '2370'
       BaseClass: '11'
        SubClass: '80'
       Interrupt: 10
   BaseAddresses: [1×6 struct]
      VendorName: 'General Standards'
         Release: 'R14SP2 or Earlier'
           Notes: 'Uses Compact PCI and PCI carriers'
      DeviceName: 'PMC-ADADIO'
      DeviceType: 'AI AO DI DO'
          ADChan: '0'
          DAChan: '4'
         DIOChan: '8'
```

Return Information for All Devices on Default Computer

Start the default target computer with the Simulink Real-Time kernel. Check 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 =

  struct with fields:

             Bus: 0
            Slot: 0
        Function: 0
        VendorID: '8086'
        DeviceID: '150'
     SubVendorID: '8086'
     SubDeviceID: '150'
       BaseClass: '6'
        SubClass: '0'
       Interrupt: 0
   BaseAddresses: [1×6 struct]
      VendorName: 'Intel'
         Release: ''
           Notes: ''
      DeviceName: 'Unknown'
      DeviceType: 'Host Bridge'
          ADChan: ''
          DAChan: ''
         DIOChan: ''
```

Return Verbose Information for All Devices Via `target_object`

Start the default target computer with the Simulink Real-Time kernel. To get the `target_object`, use `SimulinkealTime.target`. Check 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 =

  struct with fields:

             Bus: 0
            Slot: 0
        Function: 0
```

```
          VendorID: '8086'
          DeviceID: '150'
       SubVendorID: '8086'
       SubDeviceID: '150'
         BaseClass: '6'
          SubClass: '0'
         Interrupt: 0
    BaseAddresses: [1×6 struct]
       VendorName: 'Intel'
          Release: ''
            Notes: ''
       DeviceName: 'Unknown'
       DeviceType: 'Host Bridge'
           ADChan: ''
           DAChan: ''
          DIOChan: ''
```

Display Information for All Supported Devices

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                 IO333-325K-SFP (XMC-FPGA) DI DO (LVDS/...
Speedgoat                 IO333-410K-SFP (XMC-FPGA) DI DO (LVDS/...
```

Return Information for One Supported Device

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 =

  struct with fields:

       VendorID: '144A'
       DeviceID: '6208'
    SubVendorID: '-1'
    SubDeviceID: '-1'
     DeviceName: 'PCI-6208A'
     VendorName: 'ADLINK'
     DeviceType: 'AO DI DO'
         DAChan: '8'
         ADChan: '0'
        DIOChan: '4'
        Release: 'R14SP2 or Earlier'
          Notes: 'PCI-6208A features 8 current outputs with ...
                 ranges of 0-20 mA, 4-20 mA, and 5-25 mA'
```

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`

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 number of device
scalar

`Bus` and `Slot` uniquely identify a device in the target computer.

`Slot` — Slot number (PCI device number) of device
scalar

`Slot` and `Bus` uniquely identify a device in the target computer.

`Function` — PCI function number of device
scalar

`Function` uniquely identifies the function of a device 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 the devices 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 the devices 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 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 can be prefetched
`0` | `1`

Not used if `AddressSpaceIndicator` is `1` (I/O address).

- `0` — Address cannot be prefetched
- `1` — Address can be prefetched

`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 the devices 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 the devices 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.

`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.

See Also

`SimulinkRealTime.target`

Topics

“Command-Line Ethernet Card Selection by Index”
“PCI Bus I/O Devices”

Introduced in R2014a

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)
```

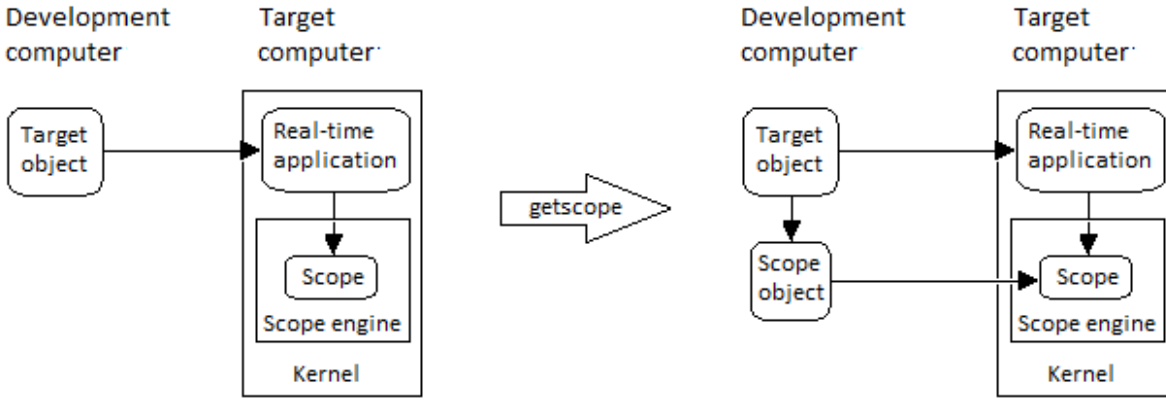
Description

`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`

`scope_number` — New scope number

`unsigned integer`

New scope number. This argument is optional. The default value is the next available integer in the target object property `Scopes`.

If you enter the scope number for an existing scope object, the result is an error.

Example: `1`

`scope_number_vector` — Vector of new scope numbers
`unsigned integer vector`

Vector of new scope numbers. If you enter the scope number for an existing scope object, the result is an error.

Example: `[2, 3]`

Output Arguments

`scope_object` — Object representing an existing scope
object

Object representing an existing scope

`scope_object_vector` — Vector of objects representing an existing scope
object

Vector containing objects representing an existing scope

See Also

`Real-Time File Scope` | `Real-Time Host Scope` | `Real-Time Target Scope` | `SimulinkRealTime.target` | `addscope` | `remscope`

Topics

“Application and Driver Scripts”

Introduced in R2014a

getsignal

Value of signal

Syntax

```
signal_value = getsignal(target_object, signal_name)

signal_value = getsignal(target_object, signal_index)

signal_values = getsignal(target_object, signal_names)
```

Description

`signal_value = getsignal(target_object, signal_name)` returns the value of signal `signal_name` at the time of the request. The value is not timestamped. 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 of the request. The value is not timestamped. Successive calls to this function do not necessarily return successive signal values.

Note Signal access by signal index will be removed in a future release. Access signals by signal name instead.

`signal_values = getsignal(target_object, signal_names)` returns the values of signals identified in the cell array of character vectors `signal_names` at the time of the request. The values are not timestamped. 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'`.

```
tg = slrt;
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
```

Get Values of Signals by Vector of Names

Get the values of signals `'Sum'` and `'Controller'`.

```
tg = slrt;
getsignal(tg, {'Sum' 'Controller'})
```

```
ans =

     0     0
```

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`

`signal_name` — Block path name of the signal

character vector | string scalar

The signal name refers to the block path of the block whose output is the specified signal. The software constructs the name according to these rules:

- If the block has more than one output port, `'/pn'` is appended to the signal name. `n` is the port number (starting at `1`).
- If the output port is not a scalar, `'/sn'` is appended to the signal name. The number `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.

For block `subsystem/path/to/block`, these rules result in the behavior listed in this table.

Output Port	Signal Name
• One output port. • The port is a scalar port.	`subsystem/path/to/block`
• One output port. • The port is a vector port. • `signal_index` refers to the second element within that vector.	`subsystem/path/to/block/s2`
• Three output ports. • The second port is a scalar port. • `signal_index` refers to the output from the second port.	`subsystem/path/to/block/p2`
• Three output ports. • The second output port is a vector port. • `signal_index` refers to the seventh element within that vector.	`subsystem/path/to/block/p2/s7`

`signal_index` — Index number of the signal
nonnegative integer

Index as shown in the `Signals` property of the real-time application. To be accessible via signal index, you must be able to observe the signal.

Note Signal access by signal index will be removed in a future release. Access signals by signal name instead.

`signal_names` — Block path names of signals
cell array of character vectors | string array

The cell array of character vectors or string array, **signal_names**, refers to the block paths of the blocks whose outputs are the specified signals. The software constructs the names according to the same rules as the **signal_name** argument.

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.

`signal_values` — Array of signal values
array of signal values

Virtual and bus signals, optimized signals, and signals of complex data types are not observable.

See Also

`SimulinkRealTime.target` | `getsignalid`

Topics

“Signal Basics” (Simulink)
“Troubleshoot Signals Not Accessible by Name”

Introduced in R2014a

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 signal name. The signal name is derived from the path to the block.

Note Signal access by signal index will be removed in a future release. Access signals by signal name instead.

For `signal_name`, enter the mangled name that the Simulink Coder software uses for code generation. To determine the mangled name:

- 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 by 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 complete names. 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 of the second element of a wide output signal 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`

`signal_name` — Block path name of the signal

character vector | string scalar

The signal name refers to the block path of the block whose output is the specified signal. The software constructs the name according to these rules:

- If the block has more than one output port, `'/pn'` is appended to the signal name. `n` is the port number (starting at `1`).
- If the output port is not a scalar, `'/sn'` is appended to the signal name. The number `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.

For block `subsystem/path/to/block`, these rules result in the behavior listed in this table.

Output Port	Signal Name
• One output port. • The port is a scalar port.	`subsystem/path/to/block`
• One output port. • The port is a vector port. • `signal_index` refers to the second element within that vector.	`subsystem/path/to/block/s2`
• Three output ports. • The second port is a scalar port. • `signal_index` refers to the output from the second port.	`subsystem/path/to/block/p2`
• Three output ports. • The second output port is a vector port. • `signal_index` refers to the seventh element within that vector.	`subsystem/path/to/block/p2/s7`

Output Arguments

`signal_index` — Index number of the signal
nonnegative integer

Index as shown in the `Signals` property of the real-time application. To be accessible via signal index, you must be able to observe the signal.

Note Signal access by signal index will be removed in a future release. Access signals by signal name instead.

See Also

`SimulinkRealTime.target` | `getsignal`

Topics

“Signal Basics” (Simulink)
“Troubleshoot Signals Not Accessible by Name”

Introduced in R2014a

getsignalidsfromlabel

Vector of signal indices

Syntax

```
index_vector = getsignalidsfromlabel(target_object, signal_label)
```

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.

Label the signal for which you request the index by using the Simulink **Signal name** parameter. You must apply 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 Signal Indices for Signal Label

Get the vector of signal indices for a signal labeled `'fourth'` in model `ex_slrt_sf_car`.

```
tg = slrt;
index_vector = getsignalidsfromlabel(tg, 'fourth')
```

```
index_vector =

     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`

`signal_label` — Label associated with a signal

character vector | string scalar

You must explicitly assign the signal label. The signal name is not a signal label.

Output Arguments

`index_vector` — Indexes into list of signals

[integer]

Vector that contains indices from the `Signals` property of the real-time application.

See Also

`SimulinkRealTime.target`

Topics

“Signal Basics” (Simulink)
“Signal Properties Controls” (Simulink)
“Troubleshoot Signals Not Accessible by Name”

Introduced in R2014a

getsignallabel

Signal label for signal index

Syntax

```
signal_label = getsignallabel(target_object, signal_index)
```

Description

`signal_label = getsignallabel(target_object, signal_index)` returns the signal label for the specified signal index, `signal_index`.

Note Signal access by signal index will be removed in a future release. Access signals by signal name instead.

Label the signal for which you request the index by using the Simulink **Signal name** parameter. The Simulink Real-Time software refers to Simulink signal names as *signal labels*.

Examples

Get Signal Label for Signal Index

Get the label for signal index 29 in model `ex_slrt_sf_car`.

```
tg = slrt;
signal_label = getsignallabel(tg, 29)
```

```
signal_label =

    'turbine torque'
```

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`

`signal_index` — Index number of the signal

nonnegative integer

Index as shown in the `Signals` property of the real-time application. To be accessible via signal index, you must be able to observe the signal.

Note Signal access by signal index will be removed in a future release. Access signals by signal name instead.

Output Arguments

`signal_label` — Label associated with a signal

character vector

You must explicitly assign the signal label. The signal name is not a signal label.

See Also

`SimulinkRealTime.target` | `getsignalidsfromlabel`

Topics

“Signal Basics” (Simulink)

“Signal Properties Controls” (Simulink)
“Troubleshoot Signals Not Accessible by Name”

Introduced in R2014a

getsignalname

Signal name from index list

Syntax

```
signal_name = getsignalname(target_object, signal_index)
```

Description

`signal_name = getsignalname(target_object, signal_index)` returns a signal name for the specified signal index.

Note Signal access by signal index will be removed in a future release. Access signals by signal name instead.

Examples

Get Signal Name from Signal Index

Get the signal name for signal index 29 in model `ex_slrt_sf_car`.

```
tg = slrt;
signal_name = getsignalname(tg,29)
```

```
signal_name =

    'transmission/Torque Converter/turbine'
```

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`

`signal_index` — Index number of the signal
nonnegative integer

Index as shown in the `Signals` property of the real-time application. To be accessible via signal index, you must be able to observe the signal.

Note Signal access by signal index will be removed in a future release. Access signals by signal name instead.

Output Arguments

`signal_name` — Block path name of the signal
character vector

The signal name refers to the block path of the block whose output is the specified signal. The software constructs the name according to these rules:

- If the block has more than one output port, `'/pn'` is appended to the signal name. `n` is the port number (starting at `1`).
- If the output port is not a scalar, `'/sn'` is appended to the signal name. The number `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.

For block `subsystem/path/to/block`, these rules result in the behavior listed in this table.

Output Port	Signal Name
• One output port. • The port is a scalar port.	`subsystem/path/to/block`

Output Port	Signal Name
• One output port. • The port is a vector port. • `signal_index` refers to the second element within that vector.	`subsystem/path/to/block/s2`
• Three output ports. • The second port is a scalar port. • `signal_index` refers to the output from the second port.	`subsystem/path/to/block/p2`
• Three output ports. • The second output port is a vector port. • `signal_index` refers to the seventh element within that vector.	`subsystem/path/to/block/p2/s7`

See Also

`SimulinkRealTime.target`

Topics

“Signal Basics” (Simulink)
“Troubleshoot Signals Not Accessible by Name”

Introduced in R2014a

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.

The maximum file size for the real-time application (`.mldatx`) file is 4 GB. This file does not include portions of the file that can be stored on the development computer (for example, a supporting `.dmr` file). Simulink Real-Time issues an error if the real-time application exceeds the maximum size or if the folder that contains the support files (`/host`) is not available.

Examples

Load Model

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`

`real_time_application` — Name of real-time application

character vector | string scalar

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` does nothing. 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` | `string`

See Also

`SimulinkRealTime.target` | `unload`

Topics

“Application and Driver Scripts”

Introduced in R2014a

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 `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

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`

`filename` — Name of a file in the target computer file system
character vector | string scalar

Enter the name of the file that contains the saved parameters.

Example: `'xpcosc_params.dat'`

Data Types: `char` | `string`

See Also

`SimulinkRealTime.target` | `saveparamset`

Introduced in R2014a

ping

Test communication between development and target computers

Syntax

```
link_status = ping(target_object)
link_status = ping(target_object, 'default')
[link_status connection_info] = ping(target_object)

[link_status connection_info] = ping(target_object, 'info')

[link_status connection_info] = ping(target_object, 'reset')
```

Description

`link_status = ping(target_object)` tests at a low level whether the development computer and the target computer represented by `target_object` can communicate using the settings stored in `target_object`. If a data channel is open between the development and target computers, the function leaves it open.

`link_status = ping(target_object, 'default')` and `[link_status connection_info] = ping(target_object)` have the same behavior as `ping(target_object)`.

`[link_status connection_info] = ping(target_object, 'info')` uses the information/control channel to return information about the Simulink Real-Time connection between the development and target computers. If a data channel is open between the development and target computers, the function leaves it open.

`[link_status connection_info] = ping(target_object, 'reset')` uses the information/control channel to close an open data channel between the development and target computers and then returns link status and connection information.

Examples

Check Communication with Responsive Target Computer

```
target_object = slrt;
link_status = ping(target_object)

link_status =

success
```

Check Communication with Unresponsive Target Computer

```
target_object = slrt('TargetPC1');
link_status = ping(target_object)

link_status =

failed
```

Get Information About Active Target Computer Connection

```
target_object = slrt;
[link_status connection_info] = ping(target_object, 'info')

link_status =

success

connection_info =

10.10.10.100
```

Get Information About Inactive Target Computer Connection

```
target_object = slrt('TargetPC1');
[link_status connection_info] = ping(target_object, 'info')

link_status =

success
```

```
connection_info =

Disconnected
```

Get Information About Unresponsive Target Computer

```
target_object = slrt('TargetPC1');
[link_status connection_info] = ping(target_object, 'info')
```

```
link_status =

failed

connection_info =

'fail: Target machine did not respond.'
```

Reset Connected Target Computer

```
target_object = slrt;
[link_status connection_info] = ping(target_object, 'reset')
```

```
link_status =

success

connection_info =

Disconnected
```

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`

Output Arguments

`link_status` — Reports if communication is possible between the development and target computers
`'success'` | `'failed'`

- If communication is possible between the development and target computers, this value is `'success'`. The value `'success'` does not mean that Simulink Real-Time has established a connection, only that one is possible.
- If communication is not possible between the development and target computers, this value is `'failed'`. The function returns `'failed'` for such reasons as a faulty or disconnected Ethernet cable or an erroneous IP address setting. For more information, see “Troubleshoot Communication Failure with Target Computers”.

`connection_info` — Reports whether a connection is active to a development computer network address
`'xx:xx:xx:xx'` | `'Disconnected'` | character vector

If you call `ping` without a second argument:

- If communication is possible, `connection_info` is empty.
- If communication is not possible, `connection_info` contains an error message.

If you call `ping` with a second argument of `'info'`:

- If the connection is active, `connection_info` reports the development computer network address to which the target computer is connected.
- If the connection is not active, `connection_info` contains `'Disconnected'`.
- If communication is not possible, `connection_info` contains an error message.

If you call `ping` with a second argument of `'reset'`:

- If communication is possible, `connection_info` contains `'Disconnected'`.
- If communication is not possible, `connection_info` contains an error message.

See Also

`SimulinkRealTime.target` | `slrtpingtarget`

Topics

“Troubleshoot Communication Failure with Target Computers”

Introduced in R2014a

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`

See Also

“Target Computer Commands” | `SimulinkRealTime.target`

Introduced in R2014a

remscope

Remove scope from target computer

Syntax

```
remscope(target_object)
remscope(target_object, scope_number)
remscope(target_object, scope_number_vector)
```

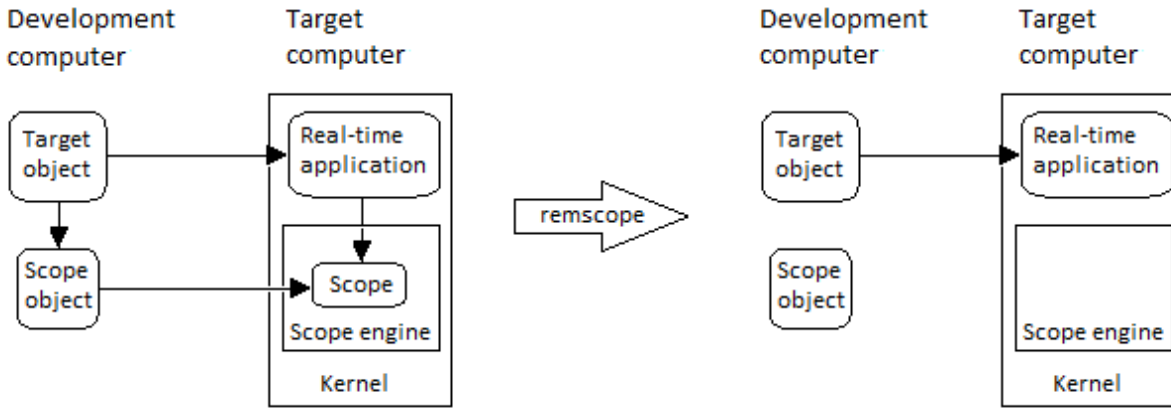
Description

`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`

`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 File Scope` | `Real-Time Host Scope` | `Real-Time Target Scope` | `SimulinkRealTime.target` | `addscope` | `getscope`

Introduced in R2014a

resetProfiler

Reset profiling service state to `Ready`

Syntax

```
resetProfiler(target_object)
```

Description

`resetProfiler(target_object)` resets the profiling service state to `Ready`, abandoning any data that the profiler has collected.

The profiler resets itself if a real-time application is loaded, started, or unloaded.

Examples

Reset Profiler

Start profiling and then reset the profiler. The real-time application is already running.

```
tg = slrt;
startProfiler(tg);
start(tg);


resetProfiler(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`

See Also

Enable Profiler | `SimulinkRealTime.ProfilerData` | `SimulinkRealTime.target`

Topics

“Execution Profiling for Real-Time Applications”

Introduced in R2017b

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

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`

`filename` — Name of a file in the target computer file system

character vector | string scalar

Enter the name of the file to receive the saved parameters.

Example: `'xpcosc_params.dat'`

Data Types: `char` | `string`

See Also

`SimulinkRealTime.target` | `loadparamset`

Introduced in R2014a

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 tunable 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 tunable global parameter to a new value. Specify the global parameter by MATLAB variable name.

`setparam(target_object, parameter_index, parameter_value)` sets the value of the tunable block or global parameter 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 tunable block or global 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 tunable block or global 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 Global Parameter by Scalar Parameter Name

Set the value of MATLAB variable `'Freq'` to `30`.

```
tg = slrt;
setparam(tg, 'Freq', 30)
```

Set Global Parameter 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 Name 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 Global Parameter by Parameter Name and Return History

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 Global Parameter by Field Name and Return History

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 Global Parameter 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`

`parameter_block_name` — Hierarchical name of the originating block

character vector | string scalar

The empty character vector (`''`) or empty string scalar (`""`) as a block name marks a global parameter 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 | string scalar

The parameter can designate either a block parameter or a global parameter that provides the value for a block parameter. To be accessible via parameter name, the block parameter or MATLAB variable 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 global parameter that provides the value for a block parameter. To be accessible via parameter index, the block parameter or MATLAB variable must be observable.

To access a parameter index, type `tg.ShowParameters = 'on'` in the Command Window, and count lines starting with `0`.

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 | string scalar | 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 `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 or string scalars, `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`

See Also
`SimulinkRealTime.target` | `getparam` | `getparamid`

Topics

“Tunable Block Parameters and Tunable Global Parameters”
“Troubleshoot Parameters Not Accessible by Name”

Introduced in R2014a

start

Starts 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 affect.

At the target computer command line, you can start execution of the real-time application by using the command `start`.

Examples

Start Execution of Real-Time Application

Start execution of the real-time application represented by the target object `tg`.

```
tg = slrt('TargetPC1');
load(tg, 'my_xpctank')
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`

See Also

“Target Computer Commands” | `SimulinkRealTime.target` | `stop`

Introduced in R2014a

startProfiler

Start profiling service on target computer

Syntax

```
startProfiler(target_object)
```

Description

`startProfiler(target_object)` starts the profiler on the target computer. Its behavior depends on the value of `ProfilerStatus`.

If `ProfilerStatus` is `Ready`:

- If a real-time application is running on the target computer, the profiler initializes and starts to collect data.
- If an application is not running, the profiler initializes and waits. When an application starts running, the profiler starts to collect data.

If `ProfilerStatus` is `DataAvailable`:

- If an application is running, calling this function returns an error. Download the data or reset the profiler before restarting it.
- If an application is not running, calling this function restarts the profiler, and this operation discards the existing profile data from the target computer.

The amount of data collected is limited to 1GB. The profiler stops by itself when it reaches this limit.

Examples

Start Profiler with Real-Time Application Running

Starts the profiler. The real-time application is already running.

```
tg = slrt;
start(tg);
startProfiler(tg);
```

Start Profiler Without Real-Time Application Running

Starts the profiler. Because the real-time application is not running, the profiler captures data from real-time application startup.

```
tg = slrt;
startProfiler(tg);
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`

See Also

Enable Profiler | `SimulinkRealTime.ProfilerData` | `SimulinkRealTime.target` | `resetProfiler` | `stopProfiler`

Topics

“Execution Profiling for Real-Time Applications”

Introduced in R2017b

stop

Stops 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 does nothing.

At the target computer command line, you can stop execution of the real-time application by using the command `stop`.

Examples

Stop Execution of Real-Time Application

Stop execution of the real-time application represented by the target object `tg`.

```
tg = slrt('TargetPC1');
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`

See Also

“Target Computer Commands” | `SimulinkRealTime.target` | `start`

Introduced in R2014a

stopProfiler

Stop profiling service on target computer

Syntax

```
stopProfiler(target_object)
```

Description

`stopProfiler(target_object)` stops the profiler from running on the target computer.

If the profiler collected data, the data is available for download to the development computer.

If the profiler did not collect data, the profiler is ready to restart.

The amount of data collected is limited to 1GB. The profiler stops by itself when it reaches this limit.

If you stop execution of the real-time application with `stop(target)`, that action also calls `stopProfiler(target)`.

Examples

Start and Stop Profiler

Starts and stops the profiler. The real-time application is already running.

```
tg = slrt;
startProfiler(tg);

stopProfiler(tg);
```

At this point, you must call either `getProfilerData` or `resetProfiler`.

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`

See Also

Enable Profiler | `SimulinkRealTime.ProfilerData` | `SimulinkRealTime.target` | `getProfilerData` | `resetProfiler`

Topics

“Execution Profiling for Real-Time Applications”

Introduced in R2017b

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 does nothing. 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`

See Also

`SimulinkRealTime.target` | `load`

Introduced in R2014a

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`

See Also

`SimulinkRealTime.target`

Introduced in R2014a

SimulinkRealTime.prototype.Instrumentation

Create a real-time instrumentation object

Description

The `SimulinkRealTime.Instrumentation` object streams signal data from a real-time simulation running on a target computer to a development computer. The instrumentation object uses the streaming API from `SimulinkRealTime.SignalList`.

Creation

`instrument_object = SimulinkRealTime.prototype.Instrumentation('mldatxFile')` creates an empty instrumentation object.

Example: “Create Instrumentation Object” on page 6-317

Properties

`AcquireList` — SimulinkRealTime.AcquireList object
object

This read-only property is updated as the instrumentation object is populated.

`nLine` — number of lines
0 (default) | integer

This read-only property is updated as the instrumentation object is populated.

Object Functions

addSignal	Adds signal for streaming to make it available in callback
clearData	Clears data from children of real-time instrumentation object

connectCallback	Adds callback that responds to new data
connectLine	Connects signal for streaming to axis
connectScalar	Adds signal for streaming to scalar display
delete	Deletes real-time instrumentation object
start	Starts streaming data from real-time application
stop	Stops streaming data from real-time application

Examples

Create Instrumentation Object

Create instrumentation object *hInst*.

```
mldatxfile = 'my_xpctank.mldatx';
hInst  = SimulinkRealTime.prototype.Instrumentation(mldatxfile);
% for technical preview in R2019b, function name includes '.prototype'
```

See Also

`addSignal` | `clearData` | `connectCallback` | `connectLine` | `connectScalar` | `delete` | `start` | `stop`

Introduced in R2019b

addSignal

Adds signal for streaming to make it available in callback

Syntax

```
addSignal(instrument_object,blockPath,portIndex,Name,Value)
addSignal(instrument_object,signalName,Name,Value)
addSignal(instrument_object,signalList,Name,Value)
```

Description

`addSignal(instrument_object,blockPath,portIndex,Name,Value)` adds a signal by using block path and port index for streaming to make the signal available in a callback. Use this approach when the signal is not used in a scalar displace or line plot.

`addSignal(instrument_object,signalName,Name,Value)` adds a signal by using signal name for streaming to make the signal available in a callback. Use this approach when the signal is not used in a scalar displace or line plot.

`addSignal(instrument_object,signalList,Name,Value)` adds signals by using signal list `SimulinkRealTime.SignalList` function for streaming to make the signals in the list available in a callback. Use this approach when the signals are not used in a scalar displace or line plot.

Examples

Add Signal by Using Block Path and Port Index

Add a signal for streaming to the real-time instrumentation object by using the block path and port index.

```
mldatxfile = 'my_xpctank.mldatx';
hInst  = SimulinkRealTime.prototype.Instrumentation(mldatxfile);
```

```
% for technical preview in R2019b, function name includes '.prototype'
addSignal(hInst,'my_xpctank/ControlValue',1);
```

Add Signal by Using Signal Name

Add a signal for streaming to the real-time instrumentation object by using the signal name.

```
% added signal name to model before building mldatxfile
mldatxfile = 'my_xpctank.mldatx';
hInst  = SimulinkRealTime.prototype.Instrumentation(mldatxfile);
% for technical preview in R2019b, function name includes '.prototype'
addSignal(hInst,'ControlValueOut');
```

Add Signals by Using Signal List

Add signals for streaming to the real-time instrumentation object by using a signal list object.

```
% added signals to signal list object after building mldatxfile
mldatxfile = 'my_xpctank.mldatx';
hInst  = SimulinkRealTime.prototype.Instrumentation(mldatxfile);
% for technical preview in R2019b, function name includes '.prototype'
mySignals = SimulinkRealTime.SignalList;
mySignals.add('my_xpctank/ControlValue',1);
mySignals.add('ControlValueOut');
addSignal(hInst,mySignals);
```

Add Signals by Signal List and Set Signal Property

This example adds signals for streaming to the real-time instrumentation object by using a signal list object. The `addSignal` function also sets a property value for the signals.

```
% added signals to signal list object after building mldatxfile
mldatxfile = 'my_xpctank.mldatx';
hInst  = SimulinkRealTime.prototype.Instrumentation(mldatxfile);
```

```
% for technical preview in R2019b, function name includes '.prototype'
addSignal(hInst,mySignals,'HistoryFlag',1);
```

Input Arguments

`instrument_object` — `SimulinkRealTime.prototype.Instrumentation` object

To create the instrumentation object, use the `SimulinkRealTime.prototype.Instrumentation` function.

Example: `hInst`

`blockPath` — Block path for the block with the signal connected to one of its outports
character vector

For the selected block, `gcb` returns the full block path name.

Example: `xpctank/ControlValue`

`portIndex` — Index of the block port that is connected to the signal for streaming
integer

For the selected signal, the output port index and signal name are visible in the signal hierarchy viewer or model data editor.

Example: 1

`signalName` — Name of the signal for streaming
character vector

For the selected signal, the port index and signal name are visible in the signal hierarchy viewer or model data editor.

Example: `ControlValueOut`

`signalList` — Signal list object that contains signals for streaming
object

To create a signal list object for a model or real-time application, use the `SimulinkRealTime.SignalList` function.

Example: `mySignals`

`Name,Value` — Name-value pairs that set properties values
name-value pair

The *`Name,Value`* pair argument selects the signal properties that are added to the instrumentation object *`instrument_object`* and sets values for the properties.

Example: `'HistoryFlag',1`

See Also

`SimulinkRealTime.prototype.Instrumentation` | `clearData` | `connectCallback` | `connectLine` | `connectScalar` | `delete` | `start` | `stop`

Introduced in R2019b

clearData

Clears data from children of real-time instrumentation object

Syntax

```
clearData(instrument_object)
```

Description

`clearData(instrument_object)` clears data from a real-time instrumentation object.

Examples

Clear Data from Instrumentation Object

Clear data from a real-time instrumentation object.

```
mldatxfile = 'my_xpctank.mldatx';
hInst  = SimulinkRealTime.prototype.Instrumentation(mldatxfile);
% for technical preview in R2019b, function name includes '.prototype'
% . . . hInst streams data
clearData(hInst);
```

Input Arguments

`instrument_object` — `SimulinkRealTime.prototype.Instrumentation` object

To create the instrumentation object, use the `SimulinkRealTime.prototype.Instrumentation` function.

Example: `hInst`

See Also

`SimulinkRealTime.prototype.Instrumentation` | `addSignal` | `connectCallback` | `connectLine` | `connectScalar` | `delete` | `start` | `stop`

Introduced in R2019b

connectCallback

Adds callback that responds to new data

Syntax

```
connectCallback(instrument_object,hCallback)
```

Description

`connectCallback(instrument_object,hCallback)` adds a callback that responds to new data that is available from the target computer. The `eventData` for the callback shares all the new data available from the target computer since the last time the callback was executed.

Examples

Add Callback for New Data Available

Add a callback that responds to new data available from the target computer and stream data to the real-time instrumentation object. This callback wraps the `XLimits` of the axes object.

```
mldatxfile = 'my_xpctank.mldatx';
hInst  = SimulinkRealTime.prototype.Instrumentation(mldatxfile);
% for technical preview in R2019b, function name includes '.prototype'
connectCallback(hInst,@my_callback);
```

Input Arguments

`instrument_object` — `SimulinkRealTime.prototype.Instrumentation` object

To create the instrumentation object, use the `SimulinkRealTime.prototype.Instrumentation` function.

Example: `hInst`

`hCallback` — a MATLAB function handle evaluated when new data is available object

The callback responds to new data becoming available for streaming.

Example: `@my_callback`

See Also

`SimulinkRealTime.prototype.Instrumentation` | `addSignal` | `clearData` | `connectLine` | `connectScalar` | `delete` | `start` | `stop`

Introduced in R2019b

connectLine

Connects signal for streaming to axis

Syntax

```
[hLine, lineName] = connectLine(instrument_object,hAxis,blockPath,
portIndex,Name,Value)
[hLine,lineName] = connectLine(instrument_object,hAxis,signalName,
Name,Value)
[hLine,lineName] = connectLine(instrument_object,hAxis,signalList,
Name,Value)
```

Description

`[hLine, lineName] = connectLine(instrument_object,hAxis,blockPath, portIndex,Name,Value)` connects a signal by using the block path and port index for streaming to an axis.

`[hLine,lineName] = connectLine(instrument_object,hAxis,signalName, Name,Value)` connects a signal by using a signal name for streaming to an axis.

`[hLine,lineName] = connectLine(instrument_object,hAxis,signalList, Name,Value)` connects a signal by using a signal list for streaming to an axis.

Examples

Connect Signal by Block Path and Port Index

Connect a signal for streaming to the real-time instrumentation object and axis object by using the block path and port index.

```
mldatxfile = 'my_xpctank.mldatx';
hInst  = SimulinkRealTime.prototype.Instrumentation(mldatxfile);
```

```
% for technical preview in R2019b, function name includes '.prototype'
[myLine,myLName1] = connectLine(hInst,myAxis,'my_xpctank/ControlValue',1);
```

Connect Signal by Signal Name

Connect a signal for streaming to the real-time instrumentation object and axis object by using a signal name.

```
% added signal name to model before building mldatxfile
mldatxfile = 'my_xpctank.mldatx';
hInst  = SimulinkRealTime.prototype.Instrumentation(mldatxfile);
% for technical preview in R2019b, function name includes '.prototype'
[myLine,myLName1] = connectLine(hInst,myAxis,'ControlValueOut');
```

Connect Signals by Signal List

Connect signals for streaming to a real-time instrumentation object and axis object by using a signal list object.

```
% added signals to signal list object after building mldatxfile
mldatxfile = 'my_xpctank.mldatx';
hInst  = SimulinkRealTime.prototype.Instrumentation(mldatxfile);
% for technical preview in R2019b, function name includes '.prototype'
[myLine,myLName1] = connectLine(hInst,myAxis,mySignals);
```

Connect Signals by Signal List and Set Line Property

Connects signals for streaming to a real-time instrumentation object and axis object by using a signal list object. The `connectLine` function in this example sets a property value for the line object `myLine`.

```
% added signals to signal list object after building mldatxfile
mldatxfile = 'my_xpctank.mldatx';
hInst  = SimulinkRealTime.prototype.Instrumentation(mldatxfile);
```

```
% for technical preview in R2019b, function name includes '.prototype'
[myLine,myLName1] = connectLine(hInst,myAxis,mySignals,'MaximumNumPoints', 5000);
```

Input Arguments

`instrument_object` — `SimulinkRealTime.prototype.Instrumentation` object

To create the instrumentation object, use the `SimulinkRealTime.prototype.Instrumentation` function.

Example: `hInst`

`hAxis` — handle to axis of a figure or UI figure
object

To create an axes object, use `hAxis = gca` or `hAxis = axes ()`.

Example: `myAxes`

`blockPath` — Block path for the block with the signal connected to one of its outports
character vector

For the selected block, `gcb` returns the full block path name.

Example: `xpctank/ControlValue`

`portIndex` — Index of the block port that is connected to the signal for streaming
integer

For the selected signal, the output port index and signal name are visible in the signal hierarchy viewer or model data editor.

Example: 1

`signalName` — Name of the signal for streaming
character vector

For the selected signal, the port index and signal name are visible in the signal hierarchy viewer or model data editor.

Example: `ControlValueOut`

`signalList` — Signal list object that contains signals for streaming
object

To create a signal list object for a model or real-time application, use the `SimulinkRealTime.SignalList` function.

Example: `mySignals`

`Name,Value` — name-value pairs that set properties values
name-value pair (default values shown) | `'MaximumNumPoints', 5000` | `'LineWidth', 2` | `'Color', [-1 -1 -1]` | `'LineStyle','-'` | `'Marker','none'` | `'MarkerSize',6`

The *`Name,Value`* pair argument selects properties and property values of the *`hLine`* object that is created when the signal is connected to the instrumentation object *`instrument_object`*.

Example: `'MaximumNumPoints', 5000`

Output Arguments

`hLine` — object representing an animated line in the `hAxis` object
object

The `animatedline` object that is created for the `hAxis` object. This line displays the streaming data from the instrumentation object.

Example: `myLine`

`lineName` — name of line in `hAxis` object
character vector

The `DisplayName` attribute that is set for the line of the `animatedline` object in the `hAxis` object.

Example: `'ControlValue line'`

See Also

`SimulinkRealTime.prototype.Instrumentation` | `addSignal` | `clearData` | `connectCallback` | `connectScalar` | `delete` | `start` | `stop`

Introduced in R2019b

connectScalar

Adds signal for streaming to scalar display

Syntax

```
hScalar = connectScalar(instrument_object,hDisplay,blockPath,
portIndex,Name,Value)
hScalar = connectScalar(instrument_object,hDisplay,signalName,
Name,Value)
hScalar = connectScalar(instrument_object,hDisplay,signalList,
Name,Value)
```

Description

`hScalar = connectScalar(instrument_object,hDisplay,blockPath, portIndex,Name,Value)` connects a signal by using the block path and port index for streaming to a scalar display as a scalar object.

`hScalar = connectScalar(instrument_object,hDisplay,signalName, Name,Value)` connects a signal by using a signal name for streaming to a scalar display as a scalar object.

`hScalar = connectScalar(instrument_object,hDisplay,signalList, Name,Value)` adds a signal by using a signal list for streaming to a scalar display as a scalar object.

Examples

Connect Signal by Using Block Path and Port Index

Connect a signal for streaming to the real-time instrumentation object and display object by using the block path and port index.

```
mldatxfile = 'my_xpctank.mldatx';
hInst  = SimulinkRealTime.prototype.Instrumentation(mldatxfile);
% for technical preview in R2019b, function name includes '.prototype'
myScalar = connectScalar(hInst,myDisplay,'my_xpctank/ControlValue',1);
```

Connect Signal by Using Signal Name

Connect a signal for streaming to the real-time instrumentation object and display object by using a signal name.

```
% added signal name to model before building mldatxfile
mldatxfile = 'my_xpctank.mldatx';
hInst  = SimulinkRealTime.prototype.Instrumentation(mldatxfile);
% for technical preview in R2019b, function name includes '.prototype'
myScalar = connectScalar(hInst,myDisplay,'ControlValueOut');
```

Connect Signals by Using Signal List

Connect signals for streaming to the real-time instrumentation object and display object by using a signal list object.

```
% added signals to signal list object after building mldatxfile
mldatxfile = 'my_xpctank.mldatx';
hInst  = SimulinkRealTime.prototype.Instrumentation(mldatxfile);
% for technical preview in R2019b, function name includes '.prototype'
myScalar = connectScalar(hInst,myDisplay,mySignals);
```

Connect Signals by Using Signal List and Scalar Property

Connect signals for streaming to the real-time instrumentation object and display object by using a signal list object. The `connectScalar` function in this example sets a property value for the scalar object `myScalar`.

```
% added signals to signal list object after building mldatxfile
mldatxfile = 'my_xpctank.mldatx';
hInst  = SimulinkRealTime.prototype.Instrumentation(mldatxfile);
```

```
% for technical preview in R2019b, function name includes '.prototype'
myScalar = connectScalar(hInst,myDisplay,mySignals,'Limits',[0 100]);
```

Input Arguments

`instrument_object` — `SimulinkRealTime.prototype.Instrumentation` object

To create the instrumentation object, use the `SimulinkRealTime.prototype.Instrumentation` function.

Example: `hInst`

`hDisplay` — Handle to a scalar display, for example an edit box, gauge, or other object

The scalar display object displays the streaming data from the instrumentation object.

Example: `myGauge`

`blockPath` — Block path for the block with the signal connected to one of its outports
character vector

For the selected block, `gcb` returns the full block path name.

Example: `xpctank/ControlValue`

`portIndex` — Index of the block port that is connected to the signal for streaming
integer

For the selected signal, the output port index and signal name are visible in the signal hierarchy viewer or model data editor.

Example: 1

`signalName` — Name of the signal for streaming
character vector

For the selected signal, the port index and signal name are visible in the signal hierarchy viewer or model data editor.

Example: `ControlValueOut`

`signalList` — Signal list object that contains signals for streaming
object

To create a signal list object for a model or real-time application, use the `SimulinkRealTime.SignalList` function.

Example: `mySignals`

`Name,Value` — Name-value pairs that set properties values
name-value pair

The *`Name,Value`* argument selects properties of the scalar object *`hScalar`* and sets values for the properties.

Example: `'Limits',[0 100]`

Output Arguments

`hScalar` — Handle to a scalar value
object

The variable object *`hScalar`* that is created for the *`hDisplay`* object. This variable displays the streaming data from the instrumentation object.

Example: `myScalar`

See Also

`SimulinkRealTime.prototype.Instrumentation` | `addSignal` | `clearData` | `connectCallback` | `connectLine` | `delete` | `start` | `stop`

Introduced in R2019b

delete

Deletes real-time instrumentation object

Syntax

```
delete(instrument_object)
```

Description

`delete(instrument_object)` deletes a real-time instrumentation object.

Examples

Delete Instrumentation Object

Delete instrumentation object `hInst`. If the instrumentation object is streaming data from a real-time application, stop streaming and delete the instrumentation object.

```
% previously . . .
% . . . created a target object
% . . . loaded/started an application on target
% . . . created an instrumentation object
% . . . optionally streamed data by using instrumentation object
delete(hInst)
```

Input Arguments

`instrument_object` — `SimulinkRealTime.prototype.Instrumentation` object

To create the instrumentation object, use the `SimulinkRealTime.prototype.Instrumentation` function.

Example: `hInst`

See Also

`SimulinkRealTime.prototype.Instrumentation` | `addSignal` | `clearData` | `connectCallback` | `connectLine` | `connectScalar` | `start` | `stop`

Introduced in R2019b

start(instrument)

Starts streaming data from real-time application

Syntax

```
start(instrument_object,target_object)
```

Description

`start(instrument_object,target_object)` validates that the real-time application for the instrumentation object is loaded on the target computer and starts streaming data from the real-time application represented by the target object. If execution of the real-time applicationis not running, this command starts the application, and then starts streaming.

Before using this method, you must:

- Create a real-time application.
- Create an instrumentation object for the real-time application, for example in a MATLAB script or AppDesigner application.
- Load the real-time application on the target computer.

Examples

Start Real-Time Application and Start Streaming Data

Start streaming data from the real-time application represented by the instrumentation object `hInst` and target object `tg`. If the application is not already running, this command starts the application, and then starts streaming data.

```
tg = slrt('TargetPC1');
load(tg, 'my_xpctank');
mldatxfile = 'my_xpctank.mldatx';
```

```
hInst  = SimulinkRealTime.prototype.Instrumentation(mldatxfile);
start(hInst,tg);
```

Start Streaming Data from Running Real-Time Application

You can start streaming data from a real-time application that is already running.

```
tg = slrt('TargetPC1');
load(tg, 'my_xpctank');
start(tg);
. . .
mldatxfile = 'my_xpctank.mldatx';
hInst  = SimulinkRealTime.prototype.Instrumentation(mldatxfile);
. . .
start(hInst,tg);
```

Input Arguments

`instrument_object` — `SimulinkRealTime.prototype.Instrumentation` object

To create the instrumentation object, use the `SimulinkRealTime.prototype.Instrumentation` function.

Example: `hInst`

`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`

See Also

`SimulinkRealTime.prototype.Instrumentation` | `clearData` | `connectCallback` | `connectLine` | `connectScalar` | `delete` | `stop`

Introduced in R2014a

stop(instrument)

Stops streaming data from real-time application

Syntax

```
stop(instrument_object)
```

Description

`stop(instrument_object)` stops streaming data from the real-time application represented by the instrumentation object. This command does not stop execution of the real-time application.

Examples

Stop Streaming Data from Real-Time Application

Stop streaming data from the real-time application represented by the instrumentation object `hInst`.

```
% previously . . .
% . . . created a target object
% . . . loaded/started an application on target
% . . . created an instrumentation object
% . . . streamed data by using instrumentation object
stop(hInst)
```

Input Arguments

`instrument_object` — `SimulinkRealTime.prototype.Instrumentation` object

To create the instrumentation object, use the `SimulinkRealTime.prototype.Instrumentation` function.

Example: `hInst`

See Also

`SimulinkRealTime.prototype.Instrumentation` | `clearData` | `connectCallback` | `connectLine` | `connectScalar` | `delete` | `start`

Introduced in R2014a

SimulinkRealTime.fileScope

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 starts collecting 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 starts collecting data into a memory buffer. It stops when it has collected the number of samples that you specified. 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 at most 128 files open on the target computer at the same time.
- The largest single file that you can create on the target computer 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 in the operating system on the target computer can have a maximum of 260 characters. If the file name is longer than eight-dot-three format (eight character file name, period, three character extension), the operating system represents the file name in truncated form (for example, six characters followed by `'~1'`). MATLAB commands can access the file using the fully qualified file name or the

truncated representation of the name. Some block parameters, such as the Scope block `filename` parameter, require 8.3 format for the file name.

- Do not write data to the `private` folder on your target computer. It is reserved for Simulink Real-Time internal use.

Before you consider using a Real-Time File Scopes that could produce a file larger than 2 GB, use either of these approaches:

- Instead of a File Scope, use the Simulation Data Inspector in buffered mode for data logging.
- If using a File Scope, use the **DynamicFileName** enabled option and set the **MaxWriteFileSize** to less than 2 GB.

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.

Creation

`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'` | character vector | string scalar

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'` | character vector | string scalar

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 | string scalar

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'` | character vector | string scalar

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 | string scalar

Read-only name of the real-time application associated with this scope object.

`ScopeId` — Unique numeric index
unsigned integer

Read-only numeric index, unique for each 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.

`Type` — Type of scope
`'Host'` (default) | `'Target'` | `'File'` | character vector | string scalar

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.

`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.

`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.

`Decimation` — Samples to acquire
`1` (default) | unsigned integer

If `1`, scope acquires every sample. If greater than `1`, scope acquires every `Decimation`th sample.

`TriggerMode` — Scope trigger mode
`'FreeRun'` (default) | `'software'` | `'signal'` | `'scope'` | character vector | string scalar

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.

`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`.

`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.

`TriggerSlope` — Trigger slope for signal trigger
`'Either'` (default) | `'Rising'` | `'Falling'` | character vector | string scalar

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.

`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.

`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.

`YLimit` — Y-axis minimum and maximum limits for scope
`'Auto'` | vector | character vector | string scalar

Select the y-axis minimum and maximum limits for scope as `Auto` or with a vector containing the minimum and maximum values (for example, `[-10, 10]`).

`Signals` — Signal indexes to display on scope
unsigned integer vector

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

Object Functions

addsignal (fileScope,signal)	Add signals to file scope represented by scope object
remsignal (fileScope,signal)	Remove signals from file scope represented by scope object
start (fileScope)	Start execution of file scope on target computer
stop (fileScope)	Stop execution of file scope on target computer
trigger(fileScope)	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 = slrt('TargetPC1');
load(tg,ex_model);
tg
```

```
Target: TargetPC1
   Connected            = Yes
   Application          = xpcosc
   Mode                 = Real-Time Single-Tasking
   Status               = stopped
   CPUOverload          = none

   ExecTime             = 0.0000
   SessionTime          = 7405.9356
   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
   ProfilerStatus       = Ready

   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
```

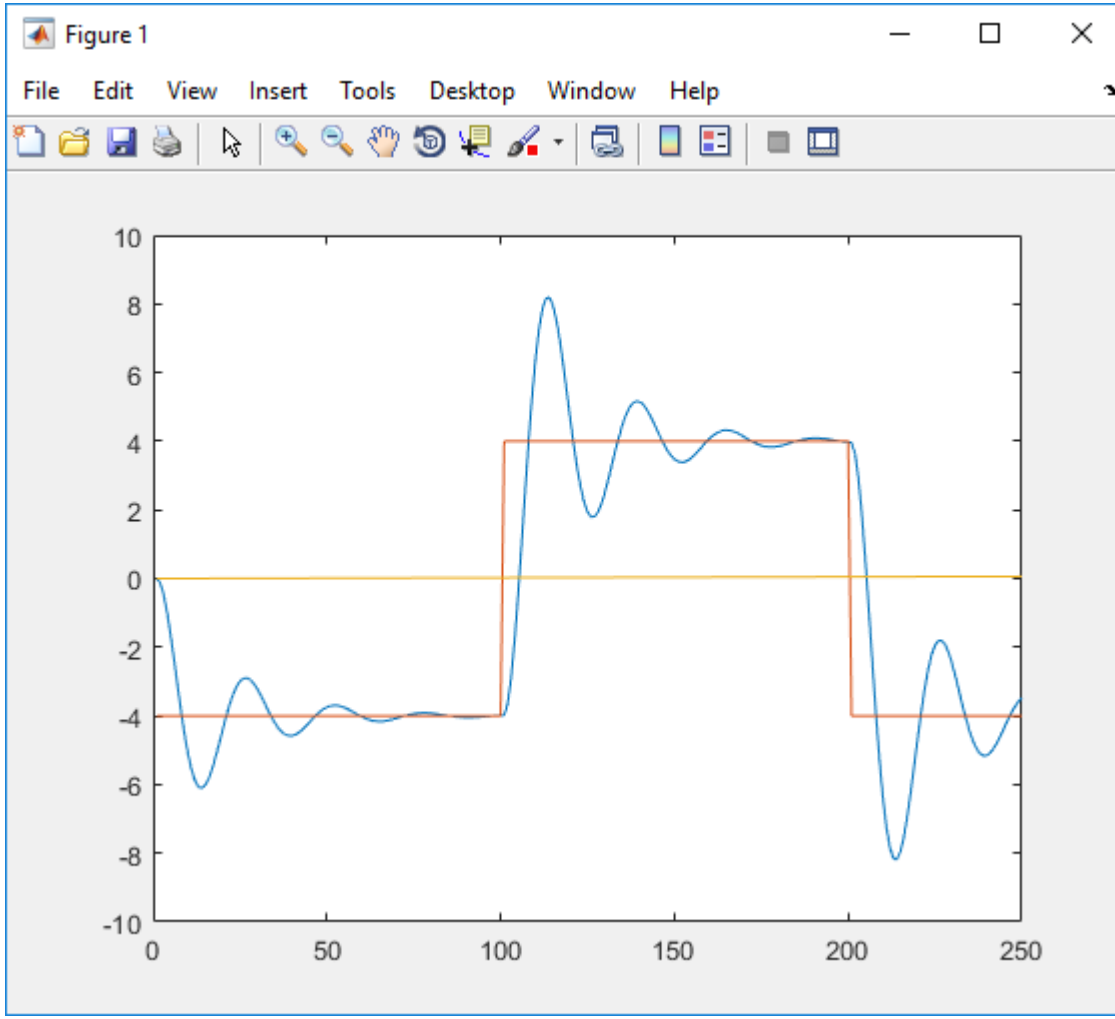
Run the real-time application for 10 seconds.

```
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” | `SimulinkRealTime.hostScope` | `SimulinkRealTime.target` | `SimulinkRealTime.targetScope` | `getscope` | `remscope`

Topics

“Simulink Real-Time Scope Usage”
“File Scope Usage”

Introduced in R2014a

addsignal (fileScope,signal)

Add signals to file scope represented by scope object

Syntax

```
scope_object_vector = addsignal(scope_object_vector,
signal_index_vector)
```

Description

`scope_object_vector = addsignal(scope_object_vector, signal_index_vector)` adds one or more signals to one or more scope objects. Specify the signals by their indices, which you can retrieve by using the target object method `SimulinkRealTime.target.getsignalid`. If `scope_object_vector` contains 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 one or more signals to the scope:

```
addsignal scope_index = signal_index1, signal_index2, . . .
```

Examples

Add Signal to a Scope

Add one signal to a file scope. The model is `xpcosc`.

Create file scope.

```
tg = slrt;
sc1 = addscope(tg, 'file', 1);
```

Get index of signal `Integrator`.

```
s1 = getsignalid(tg,'Integrator');
```

Add the signal to the scope.

```
scope_object_vector = addsignal(sc1, s1)
```

```
scope_object_vector =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 1
   Status               = Interrupted
   Type                 = File
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 1
   TriggerSample        = 0
   FileName             = unset
   WriteMode            = Lazy
   WriteSize            = 512
   AutoRestart          = off
   DynamicFileName      = off
   MaxWriteFileSize     = 536870912
   Signals              = 3  : Integrator
```

Add Signals to Multiple Scopes

Add a vector of signals to a vector of file scopes. The model is `xpcosc`.

Create file scopes.

```
tg = slrt;
sc2 = addscope(tg, 'file', 2);
sc3 = addscope(tg, 'file', 3);
```

Get indices of signals `Integrator1` and `Signal Generator`.

```
s1 = getsignalid(tg,'Integrator');
s2 = getsignalid(tg,'Signal Generator');
s3 = getsignalid(tg,'Integrator1');
```

Add the signals to the scopes.

```
scope_object_vector = addsignal([sc2 sc3], [s1, s2, s3])
```

```
scope_object_vector =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 2
   Status               = Interrupted
   Type                 = File
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 2
   TriggerSample        = 0
   FileName             = unset
   WriteMode            = Lazy
   WriteSize            = 512
   AutoRestart          = off
   DynamicFileName      = off
   MaxWriteFileSize     = 536870912
   Signals              = 3  : Integrator
                          5  : Signal Generator
                          4  : Integrator1

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 3
   Status               = Interrupted
   Type                 = File
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 3
   TriggerSample        = 0
   FileName             = unset
```

```
WriteMode            = Lazy
WriteSize            = 512
AutoRestart          = off
DynamicFileName      = off
MaxWriteFileSize     = 536870912
Signals              = 3  : Integrator
                       5  : Signal Generator
                       4  : Integrator1
```

Input Arguments

`scope_object_vector` — Vector of objects that represent scopes
scope object | [scope object]

Get a scope object by calling the target object methods `SimulinkRealTime.target.addscope` or `SimulinkRealTime.target.getscope`.

`signal_index_vector` — Vector of numbers that represent signals
unsigned integer | [unsigned integer]

Get a signal index by calling the target object method `SimulinkRealTime.target.getsignalid`.

Output Arguments

`scope_object_vector` — Vector of updated scope objects
scope object | [scope object]

This vector is the same as `scope_object_vector`, but with the changes that were made by the function call.

See Also

`SimulinkRealTime.fileScope` | `addscope` | `getscope` | `getsignalid` | `remsignal (fileScope)`

Topics

“Target Computer Commands”

“Find Signal and Parameter Indexes”
“File Scope Usage”

Introduced in R2014a

remsignal (fileScope,signal)

Remove signals from file scope represented by scope object

Syntax

```
scope_object_vector = remsignal(scope_object_vector)
scope_object_vector = remsignal(scope_object_vector,
signal_index_vector)
```

Description

`scope_object_vector = remsignal(scope_object_vector)` removes all signals from one or more scope objects. Before you can remove a signal from a scope, you must stop the scope.

`scope_object_vector = remsignal(scope_object_vector, signal_index_vector)` removes one or more signals from one or more scope objects. Specify the signals by their indices, which you can retrieve by 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.

At the target computer command line, you can remove multiple signals from the scope:

```
remsignal scope_index = signal_index1, signal_index2, . . .
```

`signal_index` is optional. If you do not include `signal_index`, all signals are removed.

Examples

Remove All Signals from One Scope

Remove all signals from scope `1`. The model is `xpcosc`.

Get the object that represents scope `1`.

```
tg = slrt;
sc1 = getscope(tg,1);
```

Remove all signals from the scope.

```
scope_object_vector = remsignal(sc1)
```

```
scope_object_vector =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 1
   Status               = Interrupted
   Type                 = File
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = -1
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 1
   TriggerSample        = 0
   FileName             = unset
   WriteMode            = Lazy
   WriteSize            = 512
   AutoRestart          = off
   DynamicFileName      = off
   MaxWriteFileSize     = 536870912
   Signals              = no Signals defined
```

Remove Selected Signals from Selected Scopes

Remove signals `'Integrator'` and `'Signal Generator'` from scopes 2 and 3.

Get the objects that represent scopes 2 and 3.

```
sc2 = getscope(tg, 2);
sc3 = getscope(tg, 3);
```

Get the signal indices that represent signals `'Integrator'` and `'Signal Generator'`.

```
s1 = getsignalid(tg,'Integrator');
s2 = getsignalid(tg,'Signal Generator');
```

Remove the signals.

```
scope_object_vector = remsignal([sc2 sc3], [s1 s2])
```

```
scope_object_vector =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 2
   Status               = Interrupted
   Type                 = File
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 2
   TriggerSample        = 0
   FileName             = unset
   WriteMode            = Lazy
   WriteSize            = 512
   AutoRestart          = off
   DynamicFileName      = off
   MaxWriteFileSize     = 536870912
   Signals              = 4  : Integrator1

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 3
   Status               = Interrupted
   Type                 = File
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 3
   TriggerSample        = 0
   FileName             = unset
   WriteMode            = Lazy
   WriteSize            = 512
```

```
AutoRestart          = off
DynamicFileName      = off
MaxWriteFileSize     = 536870912
Signals              = 4  : Integrator1
```

Input Arguments

`scope_object_vector` — Vector of objects that represent scopes
scope object | [scope object]

Get a scope object by calling the target object methods `SimulinkRealTime.target.addscope` or `SimulinkRealTime.target.getscope`.

`signal_index_vector` — Vector of numbers that represent signals
unsigned integer | [unsigned integer]

Get a signal index by calling the target object method `SimulinkRealTime.target.getsignalid`.

Output Arguments

`scope_object_vector` — Vector of updated scope objects
scope object | [scope object]

This vector is the same as `scope_object_vector`, but with the changes that were made by the function call.

See Also

`SimulinkRealTime.fileScope` | `addscope` | `addsignal (fileScope)` | `getscope` | `getsignalid`

Topics

“Target Computer Commands”
“Find Signal and Parameter Indexes”
“File Scope Usage”

Introduced in R2014a

start (fileScope)

Start execution of file scope on target computer

Syntax

```
scope_object_vector = start(scope_object_vector)
```

Description

`scope_object_vector = start(scope_object_vector)` starts one or more scopes on the target computer. Data acquisition depends on the trigger settings.

At the target computer command line, you can use the commands:

```
startscope scope_index
startscope all
```

If you use the keyword `all` at the command line, the kernel starts all of the scopes.

Examples

Start One Scope

Start scope `1`. The model is `xpcosc`.

Get the object that represents scope `1`.

```
tg = slrt;
sc1 = getscope(tg,1);
```

Start scope `1`.

```
scope_object_vector = start(sc1)
```

```
scope_object_vector =
```

```
Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 1
   Status               = Pre-Acquiring
   Type                 = File
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 1
   TriggerSample        = 0
   FileName             = c:\sc1Integ.dat
   WriteMode            = Lazy
   WriteSize            = 512
   AutoRestart          = off
   DynamicFileName      = off
   MaxWriteFileSize     = 536870912
   Signals              = 3  : Integrator
```

Start Two Scopes

Start scopes 2 and 3.

Get the objects that represent scopes 2 and 3.

```
tg = slrt;
sc2 = getscope(tg,2);
sc3 = getscope(tg,3);
```

Start the scopes.

```
scope_object_vector = start([sc2 sc3])

scope_object_vector =

 Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 2
   Status               = Pre-Acquiring
   Type                 = File
```

```
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 2
   TriggerSample        = 0
   FileName             = c:\sc2Integ.dat
   WriteMode            = Lazy
   WriteSize            = 512
   AutoRestart          = off
   DynamicFileName      = off
   MaxWriteFileSize     = 536870912
   Signals              = 3  : Integrator
                          5  : Signal Generator
                          4  : Integrator1

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 3
   Status               = Pre-Acquiring
   Type                 = File
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 3
   TriggerSample        = 0
   FileName             = c:\sc3Integ.dat
   WriteMode            = Lazy
   WriteSize            = 512
   AutoRestart          = off
   DynamicFileName      = off
   MaxWriteFileSize     = 536870912
   Signals              = 3  : Integrator
                          5  : Signal Generator
                          4  : Integrator1
```

Start All Scopes

Start all of the scopes on the target computer

Get all of the scopes.

```
tg = slrt;
allscopes = getscope(tg);
```

Start the scopes.

```
scope_object_vector = start(allscopes)
```

```
scope_object_vector =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 1
   Status               = Pre-Acquiring
   Type                 = File
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 1
   TriggerSample        = 0
   FileName             = c:\sc1Integ.dat
   WriteMode            = Lazy
   WriteSize            = 512
   AutoRestart          = off
   DynamicFileName      = off
   MaxWriteFileSize     = 536870912
   Signals              = 3  : Integrator

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 2
   Status               = Pre-Acquiring
   Type                 = File
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
```

```
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 2
   TriggerSample        = 0
   FileName             = c:\sc2Integ.dat
   WriteMode            = Lazy
   WriteSize            = 512
   AutoRestart          = off
   DynamicFileName      = off
   MaxWriteFileSize     = 536870912
   Signals              = 3  : Integrator
                          5  : Signal Generator
                          4  : Integrator1

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 3
   Status               = Pre-Acquiring
   Type                 = File
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 3
   TriggerSample        = 0
   FileName             = c:\sc3Integ.dat
   WriteMode            = Lazy
   WriteSize            = 512
   AutoRestart          = off
   DynamicFileName      = off
   MaxWriteFileSize     = 536870912
   Signals              = 3  : Integrator
```

```
5  : Signal Generator
4  : Integrator1
```

Input Arguments

`scope_object_vector` — Vector of objects that represent scopes
scope object | [scope object]

Get a scope object by calling the target object methods `SimulinkRealTime.target.addscope` or `SimulinkRealTime.target.getscope`.

Output Arguments

`scope_object_vector` — Vector of updated scope objects
scope object | [scope object]

This vector is the same as `scope_object_vector`, but with the changes that were made by the function call.

See Also

`SimulinkRealTime.fileScope` | `addscope` | `getscope` | `getsignalid` | `stop (fileScope)`

Topics

“Target Computer Commands”
“Find Signal and Parameter Indexes”
“File Scope Usage”

Introduced in R2014a

stop (fileScope)

Stop execution of file scope on target computer

Syntax

```
scope_object_vector = stop(scope_object_vector)
```

Description

`scope_object_vector = stop(scope_object_vector)` stops one or more scopes on the target computer.

At the target computer command line, you can use the commands:

```
stopscope scope_index
stopscope all
```

If you use the keyword `all` at the command line, the kernel stops all of the scopes.

Examples

Stop One Scope

Stop scope `1`. The model is `xpcosc`.

Get the object that represents scope `1`.

```
tg = slrt;
sc1 = getscope(tg,1);
```

Stop scope `1`.

```
scope_object_vector = stop(sc1)
```

```
scope_object_vector =
```

```
Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 1
   Status               = Interrupted
   Type                 = File
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 1
   TriggerSample        = 0
   FileName             = c:\sc1Integ.dat
   WriteMode            = Lazy
   WriteSize            = 512
   AutoRestart          = off
   DynamicFileName      = off
   MaxWriteFileSize     = 536870912
   Signals              = 3  : Integrator
```

Stop Two Scopes

Stop scopes 2 and 3.

Get the objects that represent scopes 2 and 3.

```
tg = slrt;
sc2 = getscope(tg,2);
sc3 = getscope(tg,3);
```

Stop the scopes.

```
scope_object_vector = stop([sc2 sc3])

scope_object_vector =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 2
   Status               = Interrupted
   Type                 = File
```

```
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 2
   TriggerSample        = 0
   FileName             = c:\sc2Integ.dat
   WriteMode            = Lazy
   WriteSize            = 512
   AutoRestart          = off
   DynamicFileName      = off
   MaxWriteFileSize     = 536870912
   Signals              = 3  : Integrator
                          5  : Signal Generator
                          4  : Integrator1

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 3
   Status               = Interrupted
   Type                 = File
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 3
   TriggerSample        = 0
   FileName             = c:\sc3Integ.dat
   WriteMode            = Lazy
   WriteSize            = 512
   AutoRestart          = off
   DynamicFileName      = off
   MaxWriteFileSize     = 536870912
   Signals              = 3  : Integrator
                          5  : Signal Generator
                          4  : Integrator1
```

Stop All Scopes

Stop all of the scopes on the target computer

Get all of the scopes.

```
tg = slrt;
allscopes = getscope(tg);
```

Stop the scopes.

```
scope_object_vector = stop(allscopes)
```

```
scope_object_vector =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 1
   Status               = Interrupted
   Type                 = File
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 1
   TriggerSample        = 0
   FileName             = c:\sc1Integ.dat
   WriteMode            = Lazy
   WriteSize            = 512
   AutoRestart          = off
   DynamicFileName      = off
   MaxWriteFileSize     = 536870912
   Signals              = 3  : Integrator

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 2
   Status               = Interrupted
   Type                 = File
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
```

```
    TriggerMode          = FreeRun
    TriggerSignal        = 3  : Integrator
    TriggerLevel         = 0.000000
    TriggerSlope         = Either
    TriggerScope         = 2
    TriggerSample        = 0
    FileName             = c:\sc2Integ.dat
    WriteMode            = Lazy
    WriteSize            = 512
    AutoRestart          = off
    DynamicFileName      = off
    MaxWriteFileSize     = 536870912
    Signals              = 3  : Integrator
                           5  : Signal Generator
                           4  : Integrator1

Simulink Real-Time Scope
    Application          = xpcosc
    ScopeId              = 3
    Status               = Interrupted
    Type                 = File
    NumSamples           = 250
    NumPrePostSamples    = 0
    Decimation           = 1
    TriggerMode          = FreeRun
    TriggerSignal        = 3  : Integrator
    TriggerLevel         = 0.000000
    TriggerSlope         = Either
    TriggerScope         = 3
    TriggerSample        = 0
    FileName             = c:\sc3Integ.dat
    WriteMode            = Lazy
    WriteSize            = 512
    AutoRestart          = off
    DynamicFileName      = off
    MaxWriteFileSize     = 536870912
    Signals              = 3  : Integrator
```

```
5  : Signal Generator
4  : Integrator1
```

Input Arguments

`scope_object_vector` — Vector of objects that represent scopes
scope object | [scope object]

Get a scope object by calling the target object methods `SimulinkRealTime.target.addscope` or `SimulinkRealTime.target.getscope`.

Output Arguments

`scope_object_vector` — Vector of updated scope objects
scope object | [scope object]

This vector is the same as `scope_object_vector`, but with the changes that were made by the function call.

See Also

`SimulinkRealTime.fileScope` | `addscope` | `getscope` | `getsignalid` | `start (fileScope)`

Topics

“Target Computer Commands”
“Find Signal and Parameter Indexes”
“File Scope Usage”

Introduced in R2014a

trigger (fileScope)

Software-trigger start of data acquisition for file scope

Syntax

```
scope_object_vector = trigger(scope_object_vector)
```

Description

`scope_object_vector = 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 the value `software` as a character vector or string scalar, this function is the only way to trigger the scope. You can use this function on any scope, regardless of trigger mode setting. For example, if a scope did not trigger because the triggering criteria were not met, you can use this function to force the scope to trigger.

Examples

Trigger Scope in Software Trigger Mode

Set a scope to software trigger mode and then force it to trigger. The model is `xpcosc`.

Set the stop time to infinity.

```
tg = slrt;
tg.StopTime = Inf;
```

Configure a scope to capture `'Integrator1'` on a software trigger.

```
sc1 = addscope(tg,'file',1);
s1 = getsignalid(tg,'Integrator1');
```

```
addsignal(sc1, s1);
sc1.TriggerMode = 'software'

sc1 =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 1
   Status               = Interrupted
   Type                 = File
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = Software
   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
```

Start the scope.

```
start(sc1)

ans =

 Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 1
   Status               = Pre-Acquiring
   Type                 = File
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = Software
   TriggerSignal        = 4  : Integrator1
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
```

```
   TriggerScope         = 1
   TriggerSample        = 0
   FileName             = c:\sc1Integ.dat
   WriteMode            = Lazy
   WriteSize            = 512
   AutoRestart          = off
   DynamicFileName      = off
   MaxWriteFileSize     = 536870912
   Signals              = 4  : Integrator1
```

Start the real-time application and trigger the scope.

```
start(tg);
pause(0.5);
trigger(sc1)

ans =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 1
   Status               = Acquiring
   Type                 = File
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = Software
   TriggerSignal        = 4  : Integrator1
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 1
   TriggerSample        = 0
   FileName             = c:\sc1Integ.dat
   WriteMode            = Lazy
   WriteSize            = 512
   AutoRestart          = off
   DynamicFileName      = off
   MaxWriteFileSize     = 536870912
   Signals              = 4  : Integrator1
```

Stop the real-time application and the scope.

```
stop(tg);
stop(sc1);
```

Input Arguments

`scope_object_vector` — Vector of objects that represent scopes
scope object | [scope object]

Get a scope object by calling the target object methods `SimulinkRealTime.target.addscope` or `SimulinkRealTime.target.getscope`.

Output Arguments

`scope_object_vector` — Vector of updated scope objects
scope object | [scope object]

This vector is the same as `scope_object_vector`, but with the changes that were made by the function call.

See Also

`SimulinkRealTime.fileScope` | `addscope` | `getscope` | `getsignalid`

Topics

“Target Computer Commands”
“Find Signal and Parameter Indexes”
“File Scope Usage”

Introduced in R2014a

SimulinkRealTime.hostScope

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.

Creation

`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 | string scalar

Read-only name of the real-time application associated with this scope object.

`ScopeId` — Unique numeric index

unsigned integer

Read-only numeric index, unique for each 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.

Type — Type of scope
`'Host'` (default) | `'Target'` | `'File'` | character vector | string scalar

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.

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.

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.

Decimation — Samples to acquire
`1` (default) | unsigned integer

If `1`, scope acquires every sample. If greater than `1`, scope acquires every `Decimation`th sample.

`TriggerMode` — Scope trigger mode

`'FreeRun'` (default) | `'software'` | `'signal'` | `'scope'` | character vector | string scalar

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.

`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`.

`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.

`TriggerSlope` — Trigger slope for signal trigger

`'Either'` (default) | `'Rising'` | `'Falling'` | character vector | string scalar

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.

`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.

`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.

`YLimit` — Y-axis minimum and maximum limits for scope
`'Auto'` | vector | character vector | string scalar

Select the y-axis minimum and maximum limits for scope as `Auto` or with a vector containing the minimum and maximum values (for example, `[-10, 10]`).

`Signals` — Signal indexes to display on scope
unsigned integer vector

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

Object Functions

addsignal (hostScope,signal)	Add signals to host scope represented by scope object
remsignal (hostScope,signal)	Remove signals from host scope represented by scope object
start (hostScope)	Start execution of host scope on target computer
stop (hostScope)	Stop execution of host scope on target computer
trigger (hostScope)	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 = slrt('TargetPC1');
load(tg,ex_model);
tg
```

```
Target: TargetPC1
   Connected            = Yes
   Application          = xpcosc
   Mode                 = Real-Time Single-Tasking
   Status               = stopped
   CPUOverload          = none

   ExecTime             = 0.0000
   SessionTime          = 7746.0916
   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
   ProfilerStatus       = Ready

   Scopes               = No Scopes defined
   NumSignals           = 7
   ShowSignals          = off
```

```
   NumParameters        = 7
   ShowParameters       = off
```

Add and configure host scope 1.

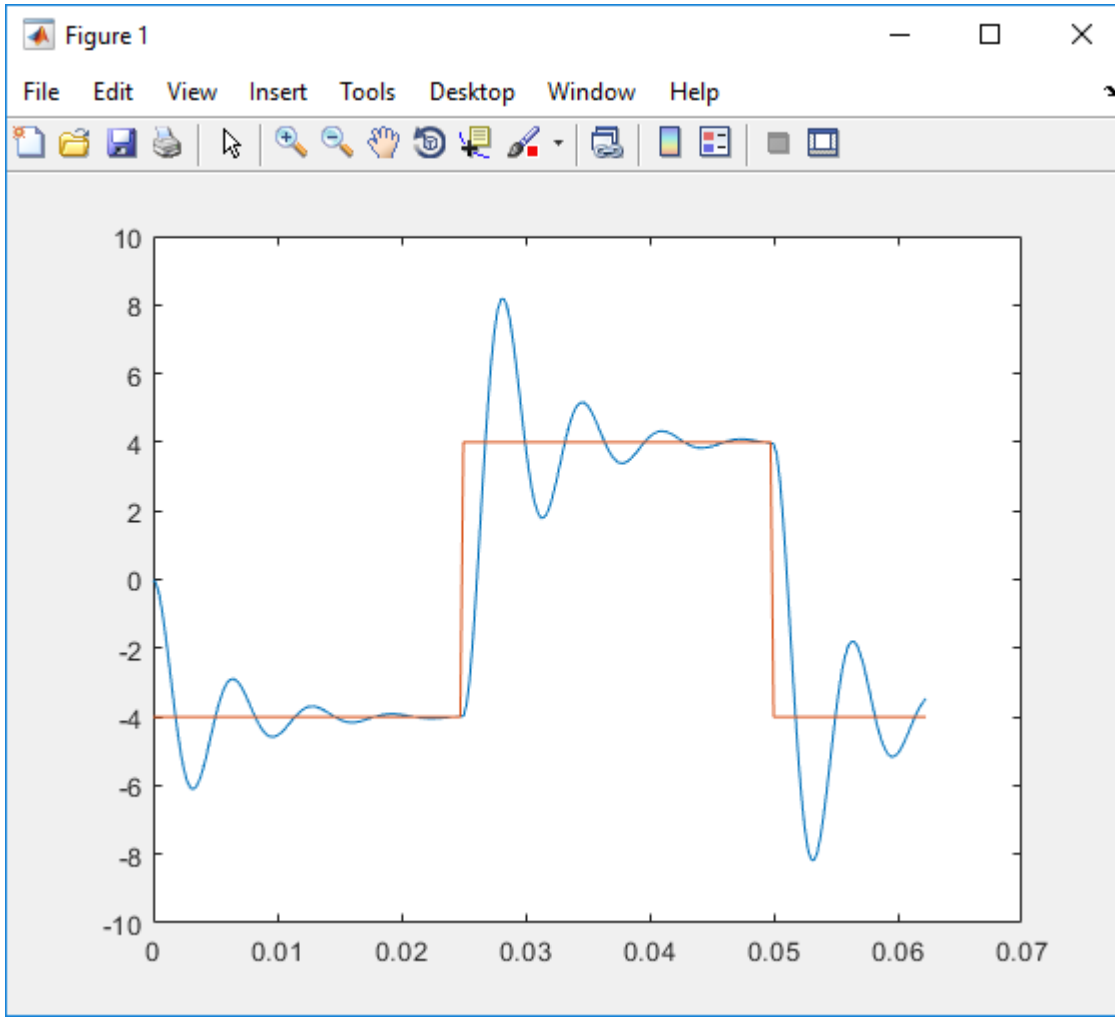
```
sc1 = addscope(tg, 'host', 1);
addsignal(sc1, 4);
addsignal(sc1, 5)

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” | `SimulinkRealTime.fileScope` | `SimulinkRealTime.target` | `SimulinkRealTime.targetScope` | `getscope` | `remscope`

Topics

“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

addsignal (hostScope,signal)

Add signals to host scope represented by scope object

Syntax

```
scope_object_vector = addsignal(scope_object_vector,
signal_index_vector)
```

Description

`scope_object_vector = addsignal(scope_object_vector, signal_index_vector)` adds one or more signals to one or more scope objects. Specify the signals by their indices, which you can retrieve by using the target object method `SimulinkRealTime.target.getsignalid`. If `scope_object_vector` contains 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 one or more signals to the scope:

```
addsignal scope_index = signal_index1, signal_index2, . . .
```

Examples

Add Signal to a Scope

Add one signal to a host scope. The model is `xpcosc`.

Create host scope.

```
tg = slrt;
sc1 = addscope(tg, 'host', 1);
```

Get index of signal `Integrator`.

```
s1 = getsignalid(tg,'Integrator');
```

Add the signal to the scope.

```
scope_object_vector = addsignal(sc1, s1)
```

```
scope_object_vector =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 1
   Status               = Interrupted
   Type                 = Host
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 1
   TriggerSample        = 0
   StartTime            = -1.000000
   Data                 = Matrix (250 x 1)
   Time                 = Matrix (250 x 1)
   Signals              = 3  : Integrator
```

Add Signals to Multiple Scopes

Add a vector of signals to a vector of host scopes. The model is `xpcosc`.

Create host scopes.

```
tg = slrt;
sc2 = addscope(tg, 'host', 2);
sc3 = addscope(tg, 'host', 3);
```

Get indices of signals `Integrator1` and `Signal Generator`.

```
s1 = getsignalid(tg,'Integrator');
s2 = getsignalid(tg,'Signal Generator');
s3 = getsignalid(tg,'Integrator1');
```

Add the signals to the scopes.

```
scope_object_vector = addsignal([sc2 sc3], [s1, s2, s3])
```

```
scope_object_vector =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 2
   Status               = Interrupted
   Type                 = Host
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 2
   TriggerSample        = 0
   StartTime            = -1.000000
   Data                 = Matrix (250 x 3)
   Time                 = Matrix (250 x 1)
   Signals              = 3  : Integrator
                          5  : Signal Generator
                          4  : Integrator1

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 3
   Status               = Interrupted
   Type                 = Host
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 3
   TriggerSample        = 0
   StartTime            = -1.000000
   Data                 = Matrix (250 x 3)
   Time                 = Matrix (250 x 1)
   Signals              = 3  : Integrator
```

```
5  : Signal Generator
4  : Integrator1
```

Input Arguments

`scope_object_vector` — Vector of objects that represent scopes
scope object | [scope object]

Get a scope object by calling the target object methods `SimulinkRealTime.target.addscope` or `SimulinkRealTime.target.getscope`.

`signal_index_vector` — Vector of numbers that represent signals
unsigned integer | [unsigned integer]

Get a signal index by calling the target object method `SimulinkRealTime.target.getsignalid`.

Output Arguments

`scope_object_vector` — Vector of updated scope objects
scope object | [scope object]

This vector is the same as `scope_object_vector`, but with the changes that were made by the function call.

See Also

`SimulinkRealTime.hostScope` | `addscope` | `getscope` | `getsignalid` | `remsignal (hostScope,signal)`

Topics

“Target Computer Commands”
“Find Signal and Parameter Indexes”
“Host Scope Usage”

Introduced in R2014a

remsignal (hostScope,signal)

Remove signals from host scope represented by scope object

Syntax

```
scope_object_vector = remsignal(scope_object_vector)
scope_object_vector = remsignal(scope_object_vector,
signal_index_vector)
```

Description

`scope_object_vector = remsignal(scope_object_vector)` removes all signals from one or more scope objects. Before you can remove a signal from a scope, you must stop the scope.

`scope_object_vector = remsignal(scope_object_vector, signal_index_vector)` removes one or more signals from one or more scope objects. Specify the signals by their indices, which you can retrieve by 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.

At the target computer command line, you can remove multiple signals from the scope:

```
remsignal scope_index = signal_index1, signal_index2, . . .
```

`signal_index` is optional. If you do not include `signal_index`, all signals are removed.

Examples

Remove All Signals from One Scope

Remove all signals from scope `1`. The model is `xpcosc`.

Get the object that represents scope `1`.

```
tg = slrt;
sc1 = getscope(tg,1);
```

Remove all signals from the scope.

```
scope_object_vector = remsignal(sc1)
```

```
scope_object_vector =

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
```

Remove Selected Signals from Selected Scopes

Remove signals `'Integrator'` and `'Signal Generator'` from scopes 2 and 3.

Get the objects that represent scopes 2 and 3.

```
sc2 = getscope(tg, 2);
sc3 = getscope(tg, 3);
```

Get the signal indices that represent signals `'Integrator'` and `'Signal Generator'`.

```
s1 = getsignalid(tg,'Integrator');
s2 = getsignalid(tg,'Signal Generator');
```

Remove the signals.

```
scope_object_vector = remsignal([sc2 sc3], [s1 s2])

scope_object_vector =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 2
   Status               = Interrupted
   Type                 = Host
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 2
   TriggerSample        = 0
   StartTime            = -1.000000
   Data                 = Matrix (250 x 1)
   Time                 = Matrix (250 x 1)
   Signals              = 4  : Integrator1

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 3
   Status               = Interrupted
   Type                 = Host
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 3
   TriggerSample        = 0
   StartTime            = -1.000000
   Data                 = Matrix (250 x 1)
```

```
Time                = Matrix (250 x 1)
Signals             = 4  : Integrator1
```

Input Arguments

`scope_object_vector` — Vector of objects that represent scopes
scope object | [scope object]

Get a scope object by calling the target object methods `SimulinkRealTime.target.addscope` or `SimulinkRealTime.target.getscope`.

`signal_index_vector` — Vector of numbers that represent signals
unsigned integer | [unsigned integer]

Get a signal index by calling the target object method `SimulinkRealTime.target.getsignalid`.

Output Arguments

`scope_object_vector` — Vector of updated scope objects
scope object | [scope object]

This vector is the same as `scope_object_vector`, but with the changes that were made by the function call.

See Also

`SimulinkRealTime.hostScope` | `addscope` | `addsignal (hostScope,signal)` | `getscope` | `getsignalid`

Topics

“Target Computer Commands”
“Find Signal and Parameter Indexes”
“Host Scope Usage”

Introduced in R2014a

start (hostScope)

Start execution of host scope on target computer

Syntax

```
scope_object_vector = start(scope_object_vector)
```

Description

`scope_object_vector = start(scope_object_vector)` starts one or more scopes on the target computer. Data acquisition depends on the trigger settings.

At the target computer command line, you can use the commands:

```
startscope scope_index
startscope all
```

If you use the keyword `all` at the command line, the kernel starts all of the scopes.

Examples

Start One Scope

Start scope `1`. The model is `xpcosc`.

Get the object that represents scope `1`.

```
tg = slrt;
sc1 = getscope(tg,1);
```

Start scope `1`.

```
scope_object_vector = start(sc1)
```

```
scope_object_vector =
```

```
Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 1
   Status               = Pre-Acquiring
   Type                 = Host
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 1
   TriggerSample        = 0
   Signals              = 3  : Integrator
```

Start Two Scopes

Start scopes 2 and 3.

Get the objects that represent scopes 2 and 3.

```
tg = slrt;
sc2 = getscope(tg,2);
sc3 = getscope(tg,3);
```

Start the scopes.

```
scope_object_vector = start([sc2 sc3])
```

```
scope_object_vector =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 2
   Status               = Pre-Acquiring
   Type                 = Host
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
```

```
   TriggerSlope         = Either
   TriggerScope         = 2
   TriggerSample        = 0
   Signals              = 3  : Integrator
                          5  : Signal Generator
                          4  : Integrator1

 Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 3
   Status               = Pre-Acquiring
   Type                 = Host
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 3
   TriggerSample        = 0
   Signals              = 3  : Integrator
                          5  : Signal Generator
                          4  : Integrator1
```

Start All Scopes

Start all of the scopes on the target computer.

Get all of the scopes.

```
tg = slrt;
allscopes = getscope(tg);
```

Start the scopes.

```
scope_object_vector = start(allscopes)

scope_object_vector =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 1
```

```
    Status               = Pre-Acquiring
    Type                 = Host
    NumSamples           = 250
    NumPrePostSamples    = 0
    Decimation           = 1
    TriggerMode          = FreeRun
    TriggerSignal        = 3  : Integrator
    TriggerLevel         = 0.000000
    TriggerSlope         = Either
    TriggerScope         = 1
    TriggerSample        = 0
    Signals              = 3  : Integrator

Simulink Real-Time Scope
    Application          = xpcosc
    ScopeId              = 2
    Status               = Pre-Acquiring
    Type                 = Host
    NumSamples           = 250
    NumPrePostSamples    = 0
    Decimation           = 1
    TriggerMode          = FreeRun
    TriggerSignal        = 3  : Integrator
    TriggerLevel         = 0.000000
    TriggerSlope         = Either
    TriggerScope         = 2
    TriggerSample        = 0
    Signals              = 3  : Integrator
                           5  : Signal Generator
                           4  : Integrator1

Simulink Real-Time Scope
    Application          = xpcosc
    ScopeId              = 3
    Status               = Pre-Acquiring
    Type                 = Host
    NumSamples           = 250
    NumPrePostSamples    = 0
    Decimation           = 1
    TriggerMode          = FreeRun
    TriggerSignal        = 3  : Integrator
    TriggerLevel         = 0.000000
    TriggerSlope         = Either
    TriggerScope         = 3
```

```
TriggerSample        = 0
Signals              = 3  : Integrator
                       5  : Signal Generator
                       4  : Integrator1
```

Input Arguments

`scope_object_vector` — Vector of objects that represent scopes
scope object | [scope object]

Get a scope object by calling the target object methods `SimulinkRealTime.target.addscope` or `SimulinkRealTime.target.getscope`.

Output Arguments

`scope_object_vector` — Vector of updated scope objects
scope object | [scope object]

This vector is the same as `scope_object_vector`, but with the changes that were made by the function call.

See Also

`SimulinkRealTime.hostScope` | `addscope` | `getscope` | `getsignalid` | `stop (hostScope)`

Topics

“Target Computer Commands”
“Find Signal and Parameter Indexes”
“Host Scope Usage”

Introduced in R2014a

stop (hostScope)

Stop execution of host scope on target computer

Syntax

```
scope_object_vector = stop(scope_object_vector)
```

Description

`scope_object_vector = stop(scope_object_vector)` stops one or more scopes on the target computer.

At the target computer command line, you can use the commands:

```
stopscope scope_index
stopscope all
```

If you use the keyword `all` at the command line, the kernel stops all of the scopes.

Examples

Stop One Scope

Stop scope `1`. The model is `xpcosc`.

Get the object that represents scope `1`.

```
tg = slrt;
sc1 = getscope(tg,1);
```

Stop scope `1`.

```
scope_object_vector = stop(sc1)
```

```
scope_object_vector =
```

```
Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 1
   Status               = Interrupted
   Type                 = Host
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 1
   TriggerSample        = 0
   StartTime            = -1.000000
   Data                 = Matrix (250 x 1)
   Time                 = Matrix (250 x 1)
   Signals              = 3  : Integrator
```

Stop Two Scopes

Stop scopes 2 and 3.

Get the objects that represent scopes 2 and 3.

```
tg = slrt;
sc2 = getscope(tg,2);
sc3 = getscope(tg,3);
```

Stop the scopes.

```
scope_object_vector = stop([sc2 sc3])
```

```
scope_object_vector =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 2
   Status               = Interrupted
   Type                 = Host
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
```

```
    TriggerMode          = FreeRun
    TriggerSignal        = 3  : Integrator
    TriggerLevel         = 0.000000
    TriggerSlope         = Either
    TriggerScope         = 2
    TriggerSample        = 0
    StartTime            = -1.000000
    Data                 = Matrix (250 x 3)
    Time                 = Matrix (250 x 1)
    Signals              = 3  : Integrator
                           5  : Signal Generator
                           4  : Integrator1

Simulink Real-Time Scope
    Application          = xpcosc
    ScopeId              = 3
    Status               = Interrupted
    Type                 = Host
    NumSamples           = 250
    NumPrePostSamples    = 0
    Decimation           = 1
    TriggerMode          = FreeRun
    TriggerSignal        = 3  : Integrator
    TriggerLevel         = 0.000000
    TriggerSlope         = Either
    TriggerScope         = 3
    TriggerSample        = 0
    StartTime            = -1.000000
    Data                 = Matrix (250 x 3)
    Time                 = Matrix (250 x 1)
    Signals              = 3  : Integrator
                           5  : Signal Generator
                           4  : Integrator1
```

Stop All Scopes

Stop all of the scopes on the target computer.

Get all of the scopes.

```
tg = slrt;
allscopes = getscope(tg);
```

Stop the scopes.

```
scope_object_vector = stop(allscopes)
```

```
scope_object_vector =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 1
   Status               = Interrupted
   Type                 = Host
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 1
   TriggerSample        = 0
   StartTime            = -1.000000
   Data                 = Matrix (250 x 1)
   Time                 = Matrix (250 x 1)
   Signals              = 3  : Integrator

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 2
   Status               = Interrupted
   Type                 = Host
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 2
   TriggerSample        = 0
   StartTime            = -1.000000
   Data                 = Matrix (250 x 3)
   Time                 = Matrix (250 x 1)
   Signals              = 3  : Integrator
                          5  : Signal Generator
                          4  : Integrator1
```

```
Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 3
   Status               = Interrupted
   Type                 = Host
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 3
   TriggerSample        = 0
   StartTime            = -1.000000
   Data                 = Matrix (250 x 3)
   Time                 = Matrix (250 x 1)
   Signals              = 3  : Integrator
                          5  : Signal Generator
                          4  : Integrator1
```

Input Arguments

`scope_object_vector` — Vector of objects that represent scopes
scope object | [scope object]

Get a scope object by calling the target object methods `SimulinkRealTime.target.addscope` or `SimulinkRealTime.target.getscope`.

Output Arguments

`scope_object_vector` — Vector of updated scope objects
scope object | [scope object]

This vector is the same as `scope_object_vector`, but with the changes that were made by the function call.

See Also

`SimulinkRealTime.hostScope` | `addscope` | `getscope` | `getsignalid` | `start (hostScope)`

Topics

“Target Computer Commands”
“Find Signal and Parameter Indexes”
“Host Scope Usage”

Introduced in R2014a

trigger (hostScope)

Software-trigger start of data acquisition for host scope

Syntax

```
scope_object_vector = trigger(scope_object_vector)
```

Description

`scope_object_vector = 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 the value `software` as a character vector or string scalar, this function is the only way to trigger the scope. You can use this function on any scope, regardless of trigger mode setting. For example, if a scope did not trigger because the triggering criteria were not met, you can use this function to force the scope to trigger.

Examples

Trigger Scope in Software Trigger Mode

Set a scope to software trigger mode and then force it to trigger. The model is `xpcosc`.

Set the stop time to infinity.

```
tg = slrt;
tg.StopTime = Inf;
```

Configure a scope to capture `'Integrator1'` on a software trigger.

```
sc1 = addscope(tg,'host',1);
s1 = getsignalid(tg,'Integrator1');
```

```
addsignal(sc1, s1);
sc1.TriggerMode = 'software'

sc1 =

   Application          = xpcosc
   ScopeId              = 1
   Status               = Interrupted
   Type                 = Host
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = Software
   TriggerSignal        = 4  : Integrator1
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 1
   TriggerSample        = 0
   StartTime            = -1.000000
   Data                 = Matrix (250 x 1)
   Time                 = Matrix (250 x 1)
   Signals              = 4  : Integrator1
```

Start the scope.

```
start(sc1)

ans =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 1
   Status               = Pre-Acquiring
   Type                 = Host
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = Software
   TriggerSignal        = 4  : Integrator1
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 1
   TriggerSample        = 0
   Signals              = 4  : Integrator1
```

Start the real-time application and trigger the scope.

```
start(tg);
pause(0.5);
trigger(sc1)

ans =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 1
   Status               = Acquiring
   Type                 = Host
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = Software
   TriggerSignal        = 4  : Integrator1
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 1
   TriggerSample        = 0
   Signals              = 4  : Integrator1
```

Stop the real-time application and the scope.

```
stop(tg);
stop(sc1);
```

Input Arguments

`scope_object_vector` — Vector of objects that represent scopes
scope object | [scope object]

Get a scope object by calling the target object methods `SimulinkRealTime.target.addscope` or `SimulinkRealTime.target.getscope`.

Output Arguments

`scope_object_vector` — Vector of updated scope objects
scope object | [scope object]

This vector is the same as `scope_object_vector`, but with the changes that were made by the function call.

See Also

`SimulinkRealTime.hostScope` | `addscope` | `getscope` | `getsignalid`

Topics

“Target Computer Commands”
“Find Signal and Parameter Indexes”
“Host Scope Usage”

Introduced in R2014a

SimulinkRealTime.ProfilerData

Contains data returned from profiler

Description

Internal format returned by profiler and displayed using public functions

The Code Execution Profiling Report displays model execution profile results for each task.

- To display the profile data for a section of the model, click the **Membrane** button next to the section.
- To display the TET data for the section in the Simulation Data Inspector, click the **Plot time series data** button .
- To view the section in Simulink Editor, click the link next to the **Expand Tree** button .
- To view the lines of generated code corresponding to the section, click the **Expand Tree** button and then click the **View Source** button .

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 lists the model sections. The numbers underneath the bars indicate the processor cores.

Creation

`getProfilerData`

Object Functions

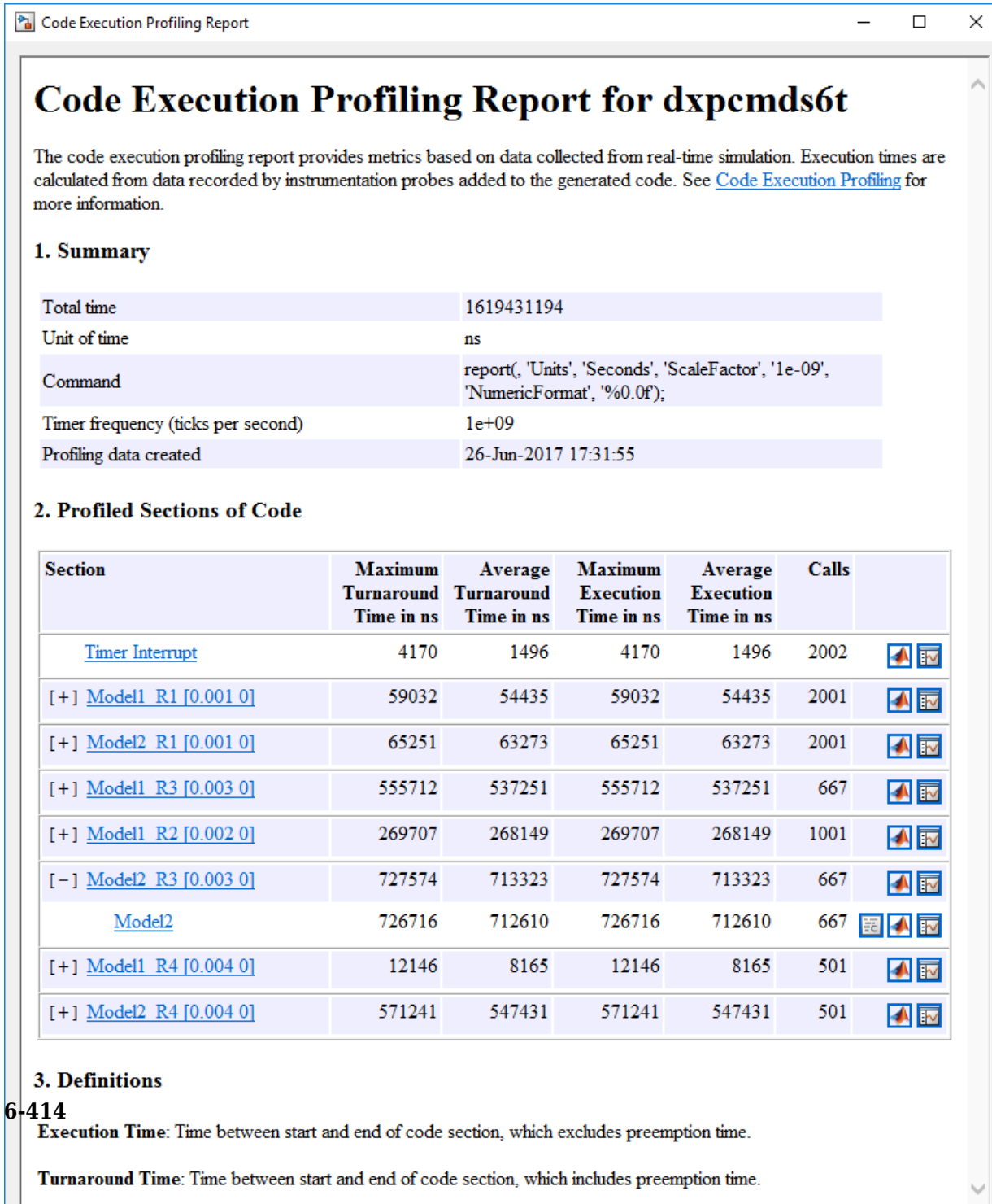
plot	Generate profiler plot
report	Generate profiler report

Examples

Run Profiler and Explicitly Display Profiler Data

Starts the profiler, stops the profiler, and retrieves results data. Calls `report` and `plot` on the results data. The real-time application `dxpcmds6t` is already loaded.

```
tg = slrt;
startProfiler(tg);
start(tg);

stopProfiler(tg);
stop(tg);

profiler_object = getProfilerData(tg);

Processing data, please wait ...

report(profiler_object);
```



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	1619431194
Unit of time	ns
Command	report(, 'Units', 'Seconds', 'ScaleFactor', '1e-09', 'NumericFormat', '%0.0f');
Timer frequency (ticks per second)	1e+09
Profiling data created	26-Jun-2017 17:31:55

2. Profiled Sections of Code

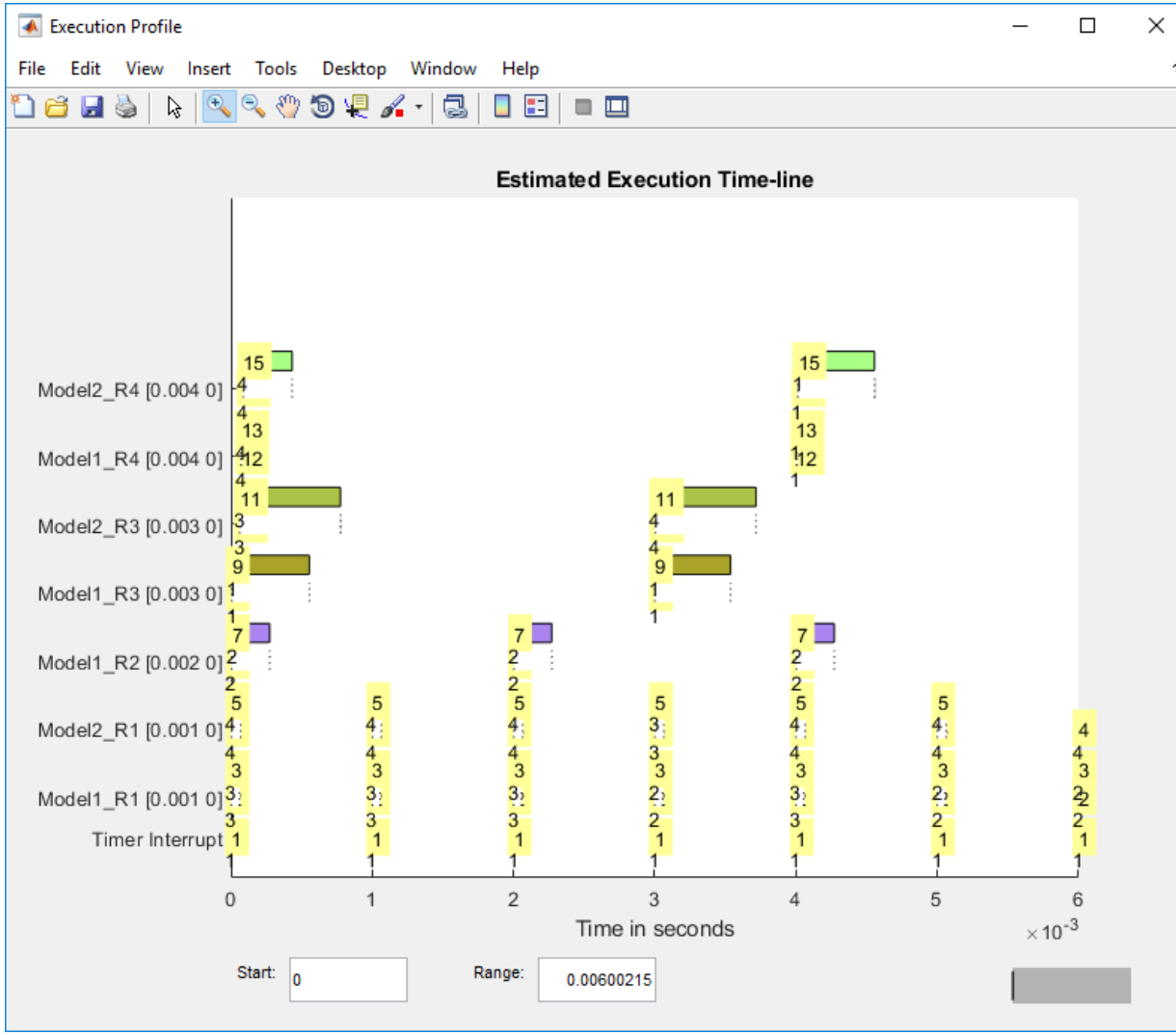
Section	Maximum Turnaround Time in ns	Average Turnaround Time in ns	Maximum Execution Time in ns	Average Execution Time in ns	Calls
Timer Interrupt	4170	1496	4170	1496	2002
[+] Model1_R1 [0.001 0]	59032	54435	59032	54435	2001
[+] Model2_R1 [0.001 0]	65251	63273	65251	63273	2001
[+] Model1_R3 [0.003 0]	555712	537251	555712	537251	667
[+] Model1_R2 [0.002 0]	269707	268149	269707	268149	1001
[−] Model2_R3 [0.003 0]	727574	713323	727574	713323	667
Model2	726716	712610	726716	712610	667
[+] Model1_R4 [0.004 0]	12146	8165	12146	8165	501
[+] Model2_R4 [0.004 0]	571241	547431	571241	547431	501

3. Definitions

Execution Time: Time between start and end of code section, which excludes preemption time.

Turnaround Time: Time between start and end of code section, which includes preemption time.

```
plot(profiler_object);
```



See Also

Enable Profiler | `getProfilerData` | `resetProfiler` | `startProfiler` | `stopProfiler`

Topics

“Execution Profiling for Real-Time Applications”

Introduced in R2017b

plot

Generate profiler plot

Syntax

```
plot(profiler_object)
```

Description

`plot(profiler_object)` generates a plot from the profiler data.

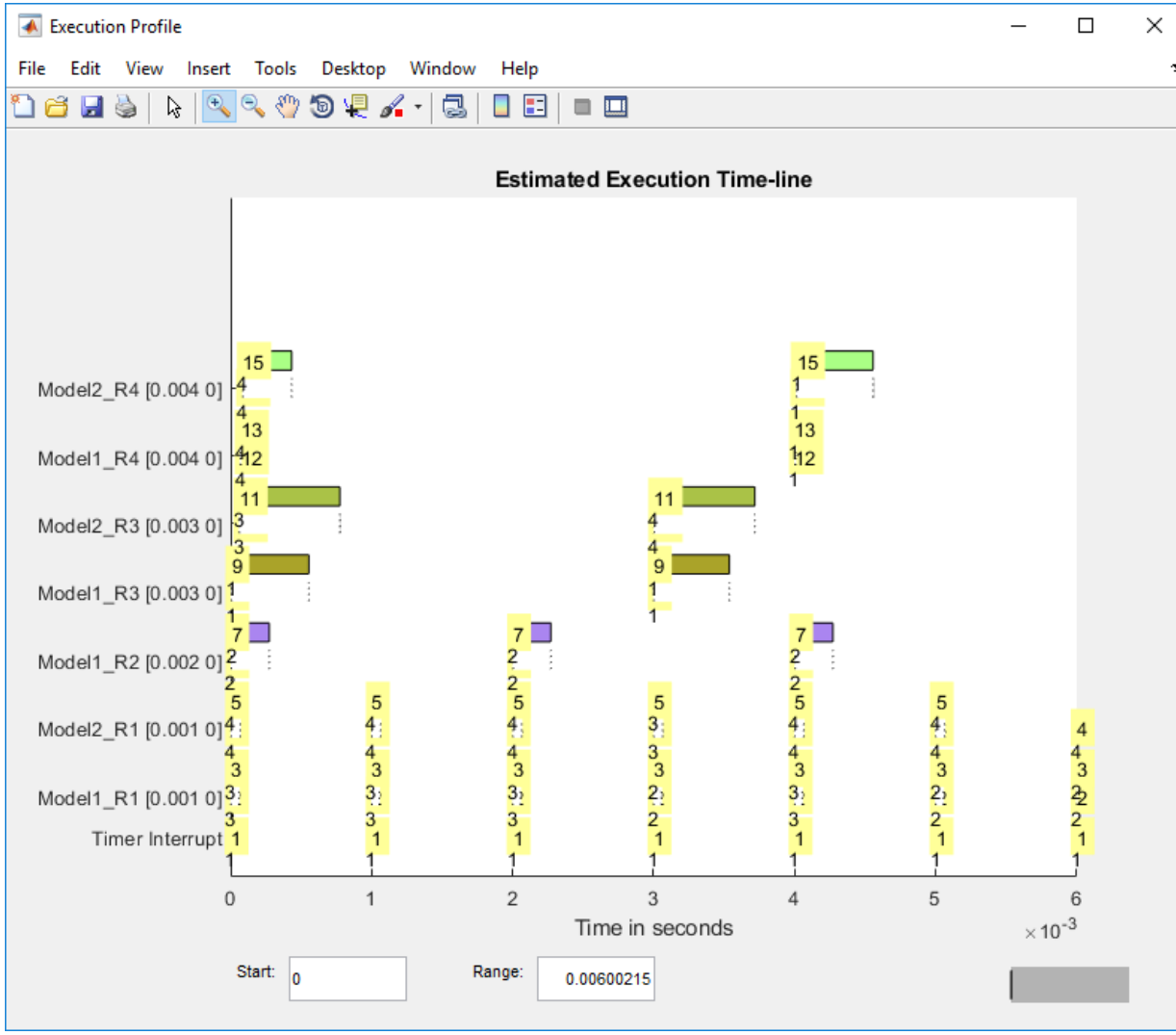
The Execution Profile plot shows the allocation of execution cycles across the four processors, indicated by the colored horizontal bars. The Code Execution Profiling Report lists the model sections. The numbers underneath the bars indicate the processor cores.

Examples

Run Profiler and Plot Profiler Data

Starts the profiler, stops the profiler, and retrieves results data. Calls `plot` on the results data. The real-time application `dxpcmds6t` is already loaded.

```
tg = slrt;
startProfiler(tg);
start(tg);

stopProfiler(tg);
stop(tg);

profiler_object = getProfilerData(tg);

Processing data, please wait ...

plot(profiler_object);
```



Input Arguments

`profiler_object` — Contains the profiler result
structure

MATLAB variable using which you can access the result of the profiler execution. You display the profiler data by calling the `plot` and `report` functions.

The structure has the following fields:

- `TargetName` — Name of target computer in target computer settings.
- `ModelInfo` — Information about model on which profiler ran:
  - `ModelName` — Name of real-time application.
  - `MATLABRelease` — MATLAB release under which model was built.
  - `KernelStamp` — Timestamp of target computer kernel build.
  - `Display` — Display mode of target computer kernel. One of `Graphics` and `Text`.
  - `BootMode` — Boot mode of target computer kernel. One of `Normal` and `Standalone`.

You can access the data in the *`profiler_object`* variable. To access the profiler data, before running the profiler, open the **Configuration Parameters**. In the **Real-Time** tab, click **Hardware Settings**. Select the **Code Generation > Verification > Workspace variable** option and set the value to `executionProfile`. Select the **Save options** option and set the value to `All data`. After running the profiler, use the technique described for the `Sections` function.

See Also

`SimulinkRealTime.ProfilerData` | `getProfilerData` | `report`

Introduced in R2017b

report

Generate profiler report

Syntax

```
report(profiler_object)
```

Description

`report(profiler_object)` generates a report from the profiler data.

The Code Execution Profiling Report displays model execution profile results for each task.

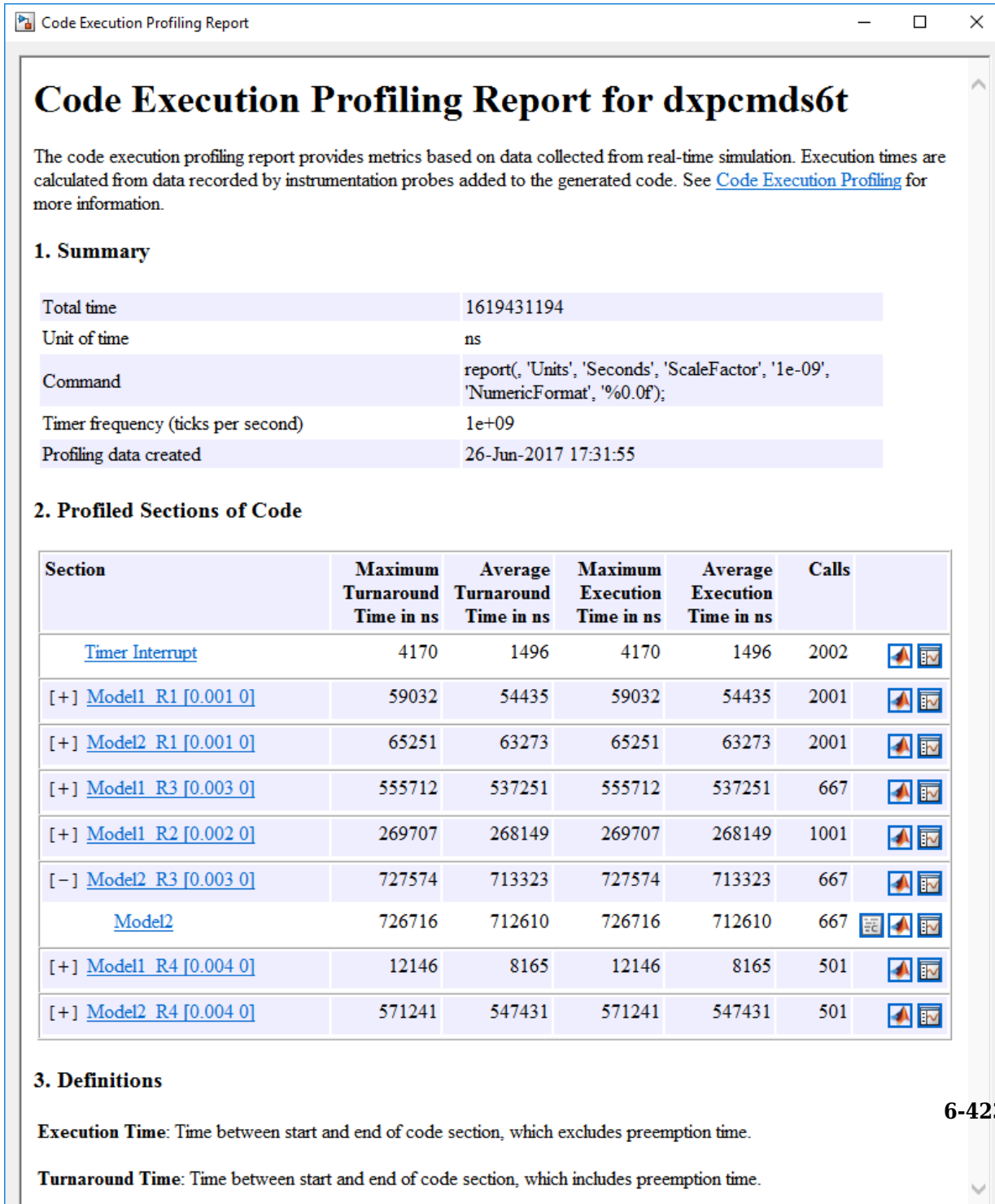
- To display the profile data for a section of the model, click the **Membrane** button next to the section.
- To display the TET data for the section in the Simulation Data Inspector, click the **Plot time series data** button .
- To view the section in Simulink Editor, click the link next to the **Expand Tree** button `[+]`.
- To view the lines of generated code corresponding to the section, click the **Expand Tree** button `[+]` and then click the **View Source** button .

Examples

Run Profiler and Report Profiler Data

Starts the profiler, stops the profiler, and retrieves results data. Calls `report` on the results data. The real-time application `dxpcmds6t` is already loaded.

```
tg = slrt;
startProfiler(tg);
start(tg);

stopProfiler(tg);
stop(tg);

profiler_object = getProfilerData(tg);

Processing data, please wait ...

report(profiler_object);
```


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	1619431194
Unit of time	ns
Command	report(, 'Units', 'Seconds', 'ScaleFactor', '1e-09', 'NumericFormat', '%0.0f');
Timer frequency (ticks per second)	1e+09
Profiling data created	26-Jun-2017 17:31:55

2. Profiled Sections of Code

Section	Maximum Turnaround Time in ns	Average Turnaround Time in ns	Maximum Execution Time in ns	Average Execution Time in ns	Calls
Timer Interrupt	4170	1496	4170	1496	2002
[+] Model1_R1 [0.001 0]	59032	54435	59032	54435	2001
[+] Model2_R1 [0.001 0]	65251	63273	65251	63273	2001
[+] Model1_R3 [0.003 0]	555712	537251	555712	537251	667
[+] Model1_R2 [0.002 0]	269707	268149	269707	268149	1001
[−] Model2_R3 [0.003 0]	727574	713323	727574	713323	667
Model2	726716	712610	726716	712610	667
[+] Model1_R4 [0.004 0]	12146	8165	12146	8165	501
[+] Model2_R4 [0.004 0]	571241	547431	571241	547431	501

3. Definitions

Execution Time: Time between start and end of code section, which excludes preemption time.

Turnaround Time: Time between start and end of code section, which includes preemption time.

MATLAB variable using which you can access the result of the profiler execution. You display the profiler data by calling the `plot` and `report` functions.

The structure has the following fields:

- `TargetName` — Name of target computer in target computer settings.
- `ModelInfo` — Information about model on which profiler ran:
  - `ModelName` — Name of real-time application.
  - `MATLABRelease` — MATLAB release under which model was built.
  - `KernelStamp` — Timestamp of target computer kernel build.
  - `Display` — Display mode of target computer kernel. One of `Graphics` and `Text`.
  - `BootMode` — Boot mode of target computer kernel. One of `Normal` and `Standalone`.

You can access the data in the *`profiler_object`* variable. To access the profiler data, before running the profiler, open the **Configuration Parameters**. In the **Real-Time** tab, click **Hardware Settings**. Select the **Code Generation > Verification > Workspace variable** option and set the value to `executionProfile`. Select the **Save options** option and set the value to `All data`. After running the profiler, use the technique described for the `Sections` function.

See Also

`SimulinkRealTime.ProfilerData` | `getProfilerData` | `plot`

Introduced in R2017b

SimulinkRealTime.targetScope

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 or rolling display.

Sliding display will be removed in a future release. It behaves like 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.

Creation

`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 | string scalar

Read-only name of the real-time application associated with this scope object.

`ScopeId` — Unique numeric index
unsigned integer

Read-only numeric index, unique for each 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.

`Type` — Type of scope
`'Host'` (default) | `'Target'` | `'File'` | character vector | string scalar

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.

`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.

`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.

`Decimation` — Samples to acquire
`1` (default) | unsigned integer

If `1`, scope acquires every sample. If greater than `1`, scope acquires every `Decimation`th sample.

`TriggerMode` — Scope trigger mode
`'FreeRun'` (default) | `'software'` | `'signal'` | `'scope'` | character vector | string scalar

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.

`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`.

`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.

`TriggerSlope` — Trigger slope for signal trigger

`'Either'` (default) | `'Rising'` | `'Falling'` | character vector | string scalar

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.

`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.

`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.

`YLimit` — Y-axis minimum and maximum limits for scope

`'Auto'` | vector | character vector | string scalar

Select the y-axis minimum and maximum limits for scope as `Auto` or with a vector containing the minimum and maximum values (for example, `[-10, 10]`).

`Signals` — Signal indexes to display on scope
unsigned integer vector

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

Object Functions

addsignal (targetScope,signal)	Add signals to target scope represented by scope object
remsignal (targetScope,signal)	Remove signals from target scope represented by scope object
start (targetScope)	Start execution of target scope on target computer
stop (targetScope)	Stop execution of target scope on target computer
trigger (targetScope)	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 = slrt('TargetPC1');
load(tg,ex_model);
tg
```

```
Target: TargetPC1
   Connected            = Yes
   Application          = xpcosc
   Mode                 = Real-Time Single-Tasking
   Status               = stopped
   CPUOverload          = none

   ExecTime             = 0.0000
   SessionTime          = 9544.6543
   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
   ProfilerStatus       = Ready

   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” | `SimulinkRealTime.fileScope` | `SimulinkRealTime.hostScope` | `SimulinkRealTime.target` | `getscope` | `remscope`

Topics

“Signal Tracing With a Target Scope”
“Simulink Real-Time Scope Usage”
“Target Scope Usage”

Introduced in R2014a

addsignal (targetScope,signal)

Add signals to target scope represented by scope object

Syntax

```
scope_object_vector = addsignal(scope_object_vector,
signal_index_vector)
```

Description

`scope_object_vector = addsignal(scope_object_vector, signal_index_vector)` adds one or more signals to one or more scope objects. Specify the signals by their indices, which you can retrieve by using the target object method `SimulinkRealTime.target.getsignalid`. If `scope_object_vector` contains 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 one or more signals to the scope:

```
addsignal scope_index = signal_index1, signal_index2, . . .
```

Examples

Add Signal to a Scope

Add one signal to a target scope. The model is `xpcosc`.

Create target scope.

```
tg = slrt;
sc1 = addscope(tg, 'target', 1);
```

Get index of signal `Integrator`.

```
s1 = getsignalid(tg,'Integrator');
```

Add the signal to the scope.

```
scope_object_vector = addsignal(sc1, s1)
```

```
scope_object_vector =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 1
   Status               = Interrupted
   Type                 = Target
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 1
   TriggerSample        = 0
   DisplayMode          = Redraw (Graphical)
   YLimit               = Auto
   Grid                 = on
   Signals              = 3  : Integrator
```

Add Signals to Multiple Scopes

Add a vector of signals to a vector of target scopes. The model is `xpcosc`.

Create target scopes.

```
tg = slrt;
sc2 = addscope(tg, 'target', 2);
sc3 = addscope(tg, 'target', 3);
```

Get indices of signals `Integrator1` and `Signal Generator`.

```
s1 = getsignalid(tg,'Integrator');
s2 = getsignalid(tg,'Signal Generator');
s3 = getsignalid(tg,'Integrator1');
```

Add the signals to the scopes.

```
scope_object_vector = addsignal([sc2 sc3], [s1, s2, s3])
```

```
scope_object_vector =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 2
   Status               = Interrupted
   Type                 = Target
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 2
   TriggerSample        = 0
   DisplayMode          = Redraw (Graphical)
   YLimit               = Auto
   Grid                 = on
   Signals              = 3  : Integrator
                          5  : Signal Generator
                          4  : Integrator1

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 3
   Status               = Interrupted
   Type                 = Target
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 3
   TriggerSample        = 0
   DisplayMode          = Redraw (Graphical)
   YLimit               = Auto
   Grid                 = on
   Signals              = 3  : Integrator
```

```
5  : Signal Generator
4  : Integrator1
```

Input Arguments

`scope_object_vector` — Vector of objects that represent scopes
scope object | [scope object]

Get a scope object by calling the target object methods `SimulinkRealTime.target.addscope` or `SimulinkRealTime.target.getscope`.

`signal_index_vector` — Vector of numbers that represent signals
unsigned integer | [unsigned integer]

Get a signal index by calling the target object method `SimulinkRealTime.target.getsignalid`.

Output Arguments

`scope_object_vector` — Vector of updated scope objects
scope object | [scope object]

This vector is the same as `scope_object_vector`, but with the changes that were made by the function call.

See Also

`SimulinkRealTime.targetScope` | `addscope` | `getscope` | `getsignalid` | `remsignal (targetScope,signal)`

Topics

“Target Computer Commands”
“Find Signal and Parameter Indexes”
“Target Scope Usage”

Introduced in R2014a

remsignal (targetScope,signal)

Remove signals from target scope represented by scope object

Syntax

```
scope_object_vector = remsignal(scope_object_vector)
scope_object_vector = remsignal(scope_object_vector,
signal_index_vector)
```

Description

`scope_object_vector = remsignal(scope_object_vector)` removes all signals from one or more scope objects. Before you can remove a signal from a scope, you must stop the scope.

`scope_object_vector = remsignal(scope_object_vector, signal_index_vector)` removes one or more signals from one or more scope objects. Specify the signals by their indices, which you can retrieve by 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.

At the target computer command line, you can remove multiple signals from the scope:

```
remsignal scope_index = signal_index1, signal_index2, . . .
```

`signal_index` is optional. If you do not include `signal_index`, all signals are removed.

Examples

Remove All Signals from One Scope

Remove all signals from scope `1`. The model is `xpcosc`.

Get the object that represents scope `1`.

```
tg = slrt;
sc1 = getscope(tg,1);
```

Remove all signals from the scope.

```
scope_object_vector = remsignal(sc1)
```

```
scope_object_vector =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 1
   Status               = Interrupted
   Type                 = Target
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = -1
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 1
   TriggerSample        = 0
   DisplayMode          = Redraw (Graphical)
   YLimit               = Auto
   Grid                 = on
   Signals              = no Signals defined
```

Remove Selected Signals from Selected Scopes

Remove signals `'Integrator'` and `'Signal Generator'` from scopes 2 and 3.

Get the objects that represent scopes 2 and 3.

```
sc2 = getscope(tg, 2);
sc3 = getscope(tg, 3);
```

Get the signal indices that represent signals `'Integrator'` and `'Signal Generator'`.

```
s1 = getsignalid(tg,'Integrator');
s2 = getsignalid(tg,'Signal Generator');
```

Remove the signals.

```
scope_object_vector = remsignal([sc2 sc3], [s1 s2])

scope_object_vector =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 2
   Status               = Interrupted
   Type                 = Target
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 2
   TriggerSample        = 0
   DisplayMode          = Redraw (Graphical)
   YLimit               = Auto
   Grid                 = on
   Signals              = 4  : Integrator1

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 3
   Status               = Interrupted
   Type                 = Target
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 3
   TriggerSample        = 0
   DisplayMode          = Redraw (Graphical)
   YLimit               = Auto
```

```
Grid                = on
Signals             = 4  : Integrator1
```

Input Arguments

`scope_object_vector` — Vector of objects that represent scopes
scope object | [scope object]

Get a scope object by calling the target object methods `SimulinkRealTime.target.addscope` or `SimulinkRealTime.target.getscope`.

`signal_index_vector` — Vector of numbers that represent signals
unsigned integer | [unsigned integer]

Get a signal index by calling the target object method `SimulinkRealTime.target.getsignalid`.

Output Arguments

`scope_object_vector` — Vector of updated scope objects
scope object | [scope object]

This vector is the same as `scope_object_vector`, but with the changes that were made by the function call.

See Also

`SimulinkRealTime.targetScope` | `addscope` | `addsignal (targetScope,signal)` | `getscope` | `getsignalid`

Topics

“Target Computer Commands”
“Find Signal and Parameter Indexes”
“Target Scope Usage”

Introduced in R2014a

start (targetScope)

Start execution of target scope on target computer

Syntax

```
scope_object_vector = start(scope_object_vector)
```

Description

`scope_object_vector = start(scope_object_vector)` starts one or more scopes on the target computer. Data acquisition depends on the trigger settings.

At the target computer command line, you can use the commands:

```
startscope scope_index
startscope all
```

If you use the keyword `all` at the command line, the kernel starts all of the scopes.

Examples

Start One Scope

Start scope `1`. The model is `xpcosc`.

Get the object that represents scope `1`.

```
tg = slrt;
sc1 = getscope(tg,1);
```

Start scope `1`.

```
scope_object_vector = start(sc1)
```

```
scope_object_vector =
```

```
Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 1
   Status               = Pre-Acquiring
   Type                 = Target
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 1
   TriggerSample        = 0
   DisplayMode          = Redraw (Graphical)
   YLimit               = Auto
   Grid                 = on
   Signals              = 3  : Integrator
```

Start Two Scopes

Start scopes 2 and 3.

Get the objects that represent scopes 2 and 3.

```
tg = slrt;
sc2 = getscope(tg,2);
sc3 = getscope(tg,3);
```

Start the scopes.

```
scope_object_vector = start([sc2 sc3])
```

```
scope_object_vector =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 2
   Status               = Pre-Acquiring
   Type                 = Target
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
```

```
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 2
   TriggerSample        = 0
   DisplayMode          = Redraw (Graphical)
   YLimit               = Auto
   Grid                 = on
   Signals              = 3  : Integrator
                          5  : Signal Generator
                          4  : Integrator1

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 3
   Status               = Pre-Acquiring
   Type                 = Target
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 3
   TriggerSample        = 0
   DisplayMode          = Redraw (Graphical)
   YLimit               = Auto
   Grid                 = on
   Signals              = 3  : Integrator
                          5  : Signal Generator
                          4  : Integrator1
```

Start All Scopes

Start all of the scopes on the target computer.

Get all of the scopes.

```
tg = slrt;
allscopes = getscope(tg);
```

Start the scopes.

```
scope_object_vector = start(allscopes)
```

```
scope_object_vector =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 1
   Status               = Pre-Acquiring
   Type                 = Target
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 1
   TriggerSample        = 0
   DisplayMode          = Redraw (Graphical)
   YLimit               = Auto
   Grid                 = on
   Signals              = 3  : Integrator

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 2
   Status               = Pre-Acquiring
   Type                 = Target
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 2
   TriggerSample        = 0
   DisplayMode          = Redraw (Graphical)
   YLimit               = Auto
   Grid                 = on
   Signals              = 3  : Integrator
                          5  : Signal Generator
                          4  : Integrator1
```

```
Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 3
   Status               = Pre-Acquiring
   Type                 = Target
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 3
   TriggerSample        = 0
   DisplayMode          = Redraw (Graphical)
   YLimit               = Auto
   Grid                 = on
   Signals              = 3  : Integrator
                          5  : Signal Generator
                          4  : Integrator1
```

Input Arguments

`scope_object_vector` — Vector of objects that represent scopes
scope object | [scope object]

Get a scope object by calling the target object methods `SimulinkRealTime.target.addscope` or `SimulinkRealTime.target.getscope`.

Output Arguments

`scope_object_vector` — Vector of updated scope objects
scope object | [scope object]

This vector is the same as `scope_object_vector`, but with the changes that were made by the function call.

See Also

`SimulinkRealTime.targetScope` | `addscope` | `getscope` | `getsignalid` | `stop (targetScope)`

Topics

“Target Computer Commands”
“Find Signal and Parameter Indexes”
“Target Scope Usage”

Introduced in R2014a

stop (targetScope)

Stop execution of target scope on target computer

Syntax

```
scope_object_vector = stop(scope_object_vector)
```

Description

`scope_object_vector = stop(scope_object_vector)` stops one or more scopes on the target computer.

At the target computer command line, you can use the commands:

```
stopscope scope_index
stopscope all
```

If you use the keyword `all` at the command line, the kernel stops all of the scopes.

Examples

Stop One Scope

Stop scope `1`. The model is `xpcosc`.

Get the object that represents scope `1`.

```
tg = slrt;
sc1 = getscope(tg,1);
```

Stop scope `1`.

```
scope_object_vector = stop(sc1)
```

```
scope_object_vector =
```

```
Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 1
   Status               = Interrupted
   Type                 = Target
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 1
   TriggerSample        = 0
   DisplayMode          = Redraw (Graphical)
   YLimit               = Auto
   Grid                 = on
   Signals              = 3  : Integrator
```

Stop Two Scopes

Stop scopes 2 and 3.

Get the objects that represent scopes 2 and 3.

```
tg = slrt;
sc2 = getscope(tg,2);
sc3 = getscope(tg,3);
```

Stop the scopes.

```
scope_object_vector = stop([sc2 sc3])
```

```
scope_object_vector =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 2
   Status               = Interrupted
   Type                 = Target
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
```

```
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 2
   TriggerSample        = 0
   DisplayMode          = Redraw (Graphical)
   YLimit               = Auto
   Grid                 = on
   Signals              = 3  : Integrator
                          5  : Signal Generator
                          4  : Integrator1

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 3
   Status               = Interrupted
   Type                 = Target
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 3
   TriggerSample        = 0
   DisplayMode          = Redraw (Graphical)
   YLimit               = Auto
   Grid                 = on
   Signals              = 3  : Integrator
                          5  : Signal Generator
                          4  : Integrator1
```

Stop All Scopes

Stop all of the scopes on the target computer.

Get all of the scopes.

```
tg = slrt;
allscopes = getscope(tg);
```

Stop the scopes.

```
scope_object_vector = stop(allscopes)
```

```
scope_object_vector =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 1
   Status               = Interrupted
   Type                 = Target
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 1
   TriggerSample        = 0
   DisplayMode          = Redraw (Graphical)
   YLimit               = Auto
   Grid                 = on
   Signals              = 3  : Integrator

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 2
   Status               = Interrupted
   Type                 = Target
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 2
   TriggerSample        = 0
   DisplayMode          = Redraw (Graphical)
   YLimit               = Auto
   Grid                 = on
   Signals              = 3  : Integrator
                          5  : Signal Generator
                          4  : Integrator1
```

```
Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 3
   Status               = Interrupted
   Type                 = Target
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = FreeRun
   TriggerSignal        = 3  : Integrator
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 3
   TriggerSample        = 0
   DisplayMode          = Redraw (Graphical)
   YLimit               = Auto
   Grid                 = on
   Signals              = 3  : Integrator
                          5  : Signal Generator
                          4  : Integrator1
```

Input Arguments

`scope_object_vector` — Vector of objects that represent scopes

scope object | [scope object]

Get a scope object by calling the target object methods `SimulinkRealTime.target.addscope` or `SimulinkRealTime.target.getscope`.

Output Arguments

`scope_object_vector` — Vector of updated scope objects

scope object | [scope object]

This vector is the same as `scope_object_vector`, but with the changes that were made by the function call.

See Also

`SimulinkRealTime.targetScope` | `addscope` | `getscope` | `getsignalid` | `start (targetScope)`

Topics

“Target Computer Commands”
“Find Signal and Parameter Indexes”
“Target Scope Usage”

Introduced in R2014a

trigger (targetScope)

Software-trigger start of data acquisition for target scope

Syntax

```
scope_object_vector = trigger(scope_object_vector)
```

Description

`scope_object_vector = 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 the value `software` as a character vector or string scalar, this function is the only way to trigger the scope. You can use this function on any scope, regardless of trigger mode setting. For example, if a scope did not trigger because the triggering criteria were not met, you can use this function to force the scope to trigger.

Examples

Trigger Scope in Software Trigger Mode

Set a scope to software trigger mode and then force it to trigger. The model is `xpcosc`.

Set the stop time to infinity.

```
tg = slrt;
tg.StopTime = Inf;
```

Configure a scope to capture `'Integrator1'` on a software trigger.

```
sc1 = addscope(tg,'target',1);
s1 = getsignalid(tg,'Integrator1');
```

```
addsignal(sc1, s1);
sc1.TriggerMode = 'software'

sc1 =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 1
   Status               = Interrupted
   Type                 = Target
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = Software
   TriggerSignal        = 4  : Integrator1
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 1
   TriggerSample        = 0
   DisplayMode          = Redraw (Graphical)
   YLimit               = Auto
   Grid                 = on
   Signals              = 4  : Integrator1
```

Start the scope.

```
start(sc1)

ans =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 1
   Status               = Pre-Acquiring
   Type                 = Target
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = Software
   TriggerSignal        = 4  : Integrator1
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 1
   TriggerSample        = 0
   DisplayMode          = Redraw (Graphical)
```

```
   YLimit               = Auto
   Grid                 = on
   Signals              = 4  : Integrator1
```

Start the real-time application and trigger the scope.

```
start(tg);
pause(0.5);
trigger(sc1)

ans =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 1
   Status               = Acquiring
   Type                 = Target
   NumSamples           = 250
   NumPrePostSamples    = 0
   Decimation           = 1
   TriggerMode          = Software
   TriggerSignal        = 4  : Integrator1
   TriggerLevel         = 0.000000
   TriggerSlope         = Either
   TriggerScope         = 1
   TriggerSample        = 0
   DisplayMode          = Redraw (Graphical)
   YLimit               = Auto
   Grid                 = on
   Signals              = 4  : Integrator1
```

Stop the real-time application and the scope.

```
stop(tg);
stop(sc1);
```

Input Arguments

`scope_object_vector` — Vector of objects that represent scopes

scope object | [scope object]

Get a scope object by calling the target object methods `SimulinkRealTime.target.addscope` or `SimulinkRealTime.target.getscope`.

Output Arguments

`scope_object_vector` — Vector of updated scope objects
scope object | [scope object]

This vector is the same as `scope_object_vector`, but with the changes that were made by the function call.

See Also

`addscope` | `getscope` | `getsignalid`

Topics

“Target Computer Commands”
“Find Signal and Parameter Indexes”
“Target Scope Usage”

Introduced in R2014a